Artificial Intelligence

A Modern Approach

Stuart J. Russell and Peter Norvig

Contributing writers:
John F. Canny, Jitendra M. Malik, Douglas D. Edwards

Prentice Hall, Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Russell, Stuart J. (Stuart Jonathan)
Artificial intelligence : a modern approach / Stuart Russell, Peter Norvig.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-103805-2
1. Artificial intelligence I. Norvig, Peter. II. Title.
Q335.R86 1995
006.3--dc20 94-36444
CIP

Publisher: Alan Apt
Production Editor: Mona Pompili
Developmental Editor: Sondra Chavez
Cover Designers: Stuart Russell and Peter Norvig
Production Coordinator: Lori Bulwin
Editorial Assistant: Shirley McGuire

© 1995 by Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

All rights reserved. No part of this book may be
reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-103805-2

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada, Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Preface

There are many textbooks that offer an introduction to artificial intelligence (AI). This text has five principal features that together distinguish it from other texts.

1. *Unified presentation of the field.*
   Some texts are organized from a historical perspective, describing each of the major problems and solutions that have been uncovered in 40 years of AI research. Although there is value to this perspective, the result is to give the impression of a dozen or so barely related subfields, each with its own techniques and problems. We have chosen to present AI as a unified field, working on a common problem in various guises. This has entailed some reinterpretation of past research, showing how it fits within a common framework and how it relates to other work that was historically separate. It has also led us to include material not normally covered in AI texts.
2. *Intelligent agent design.*
   The unifying theme of the book is the concept of an *intelligent agent.* In this view, the problem of AI is to describe and build agents that receive percepts from the environment and perform actions. Each such agent is implemented by a function that maps percepts to actions, and we cover different ways to represent these functions, such as production systems, reactive agents, logical planners, neural networks, and decision-theoretic systems. We explain the role of learning as extending the reach of the designer into unknown environments, and show how it constrains agent design, favoring explicit knowledge representation and reasoning. We treat robotics and vision not as independently defined problems, but as occurring in the service of goal achievement. We stress the importance of the task environment characteristics in determining the appropriate agent design.
3. *Comprehensive and up-to-date coverage.*
   We cover areas that are sometimes underemphasized, including reasoning under uncertainty, learning, neural networks, natural language, vision, robotics, and philosophical foundations. We cover many of the more recent ideas in the field, including simulated annealing, memory-bounded search, global ontologies, dynamic and adaptive probabilistic (Bayesian) networks, computational learning theory, and reinforcement learning. We also provide extensive notes and references on the historical sources and current literature for the main ideas in each chapter.
4. *Equal emphasis on theory and practice.*
   Theory and practice are given equal emphasis. All material is grounded in first principles with rigorous theoretical analysis where appropriate, but the point of the theory is to get the concepts across and explain how they are used in actual, fielded systems. The reader of this book will come away with an appreciation for the basic concepts and mathematical methods of AI, and also with an idea of what can and cannot be done with today's technology, at what cost, and using what techniques.
5. *Understanding through implementation.*
   The principles of intelligent agent design are clarified by using them to actually build agents. Chapter 2 provides an overview of agent design, including a basic agent and environment

project. Subsequent chapters include programming exercises that ask the student to add capabilities to the agent, making it behave more and more interestingly and (we hope) intelligently. Algorithms are presented at three levels of detail: prose descriptions and pseudo-code in the text, and complete Common Lisp programs available on the Internet or on floppy disk. All the agent programs are interoperable and work in a uniform framework for simulated environments.

This book is primarily intended for use in an undergraduate course or course sequence. It can also be used in a graduate-level course (perhaps with the addition of some of the primary sources suggested in the bibliographical notes). Because of its comprehensive coverage and the large number of detailed algorithms, it is useful as a primary reference volume for AI graduate students and professionals wishing to branch out beyond their own subfield. We also hope that AI researchers could benefit from thinking about the unifying approach we advocate.

The only prerequisite is familiarity with basic concepts of computer science (algorithms, data structures, complexity) at a sophomore level. Freshman calculus is useful for understanding neural networks and adaptive probabilistic networks in detail. Some experience with nonnumeric programming is desirable, but can be picked up in a few weeks study. We provide implementations of all algorithms in Common Lisp (see Appendix B), but other languages such as Scheme, Prolog, Smalltalk, C++, or ML could be used instead.

Overview of the book

The book is divided into eight parts. Part I, "Artificial Intelligence," sets the stage for all the others, and offers a view of the AI enterprise based around the idea of intelligent agents—systems that can decide what to do and do it. Part II, "Problem Solving," concentrates on methods for deciding what to do when one needs to think ahead several steps, for example in navigating across country or playing chess. Part III, "Knowledge and Reasoning," discusses ways to represent knowledge about the world—how it works, what it is currently like, what one's actions might do—and how to reason logically with that knowledge. Part IV, "Acting Logically," then discusses how to use these reasoning methods to decide what to do, particularly by constructing *plans.* Part V, "Uncertain Knowledge and Reasoning," is analogous to Parts III and IV, but it concentrates on reasoning and decision-making in the presence of *uncertainty* about the world, as might be faced, for example, by a system for medical diagnosis and treatment.

Together, Parts II to V describe that part of the intelligent agent responsible for reaching decisions. Part VI, "Learning," describes methods for generating the knowledge required by these decision-making components; it also introduces a new kind of component, the *neural network,* and its associated learning procedures. Part VII, "Communicating, Perceiving, and Acting," describes ways in which an intelligent agent can perceive its environment so as to know what is going on, whether by vision, touch, hearing, or understanding language; and ways in which it can turn its plans into real actions, either as robot motion or as natural language utterances. Finally, Part VIII, "Conclusions," analyses the past and future of AI, and provides some light amusement by discussing what AI really is and why it has already succeeded to some degree, and airing the views of those philosophers who believe that AI can never succeed at all.

Using this book

This is a big book; covering *all* the chapters and the projects would take two semesters. You will notice that the book is divided into 27 chapters, which makes it easy to select the appropriate material for any chosen course of study. Each chapter can be covered in approximately one week. Some reasonable choices for a variety of quarter and semester courses are as follows:

- *One-quarter general introductory course:*
  Chapters 1, 2, 3, 6, 7, 9, 11, 14, 15, 18, 22.
- *One-semester general introductory course:*
  Chapters 1, 2, 3, 4, 6, 7, 9, 11, 13, 14, 15, 18, 19, 22, 24, 26, 27.
- *One-quarter course with concentration on search and planning:*
  Chapters 1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13.
- *One-quarter course with concentration on reasoning and expert systems:*
  Chapters 1, 2, 3, 6, 7, 8, 9, 10, 11, 14, 15, 16.
- *One-quarter course with concentration on natural language:*
  Chapters 1, 2, 3, 6, 7, 8, 9, 14, 15, 22, 23, 26, 27.
- *One-semester course with concentration on learning and neural networks:*
  Chapters 1, 2, 3, 4, 6, 7, 9, 14, 15, 16, 17, 18, 19, 20, 21.
- *One-semester course with concentration on vision and robotics:*
  Chapters 1, 2, 3, 4, 6, 7, 11, 13, 14, 15, 16, 17, 24, 25, 20.

These sequences could be used for both undergraduate and graduate courses. The relevant parts of the book could also be used to provide the first phase of graduate specialty courses. For example, Part VI could be used in conjunction with readings from the literature in a course on machine learning.

We have decided *not* to designate certain sections as "optional" or certain exercises as "difficult," as individual tastes and backgrounds vary widely. Exercises requiring significant programming are marked with a keyboard icon, and those requiring some investigation of the literature are marked with a book icon. Altogether, over 300 exercises are included. Some of them are large enough to be considered term projects. Many of the exercises can best be solved by taking advantage of the code repository, which is described in Appendix B. Throughout the book, important points are marked with a *pointing icon.*

If you have any comments on the book, we'd like to hear from you. Appendix B includes information on how to contact us.

Acknowledgements

Jitendra Malik wrote most of Chapter 24 (Vision) and John Canny wrote most of Chapter 25 (Robotics). Doug Edwards researched the Historical Notes sections for all chapters and wrote much of them. Tim Huang helped with formatting of the diagrams and algorithms. Maryann Simmons prepared the 3-D model from which the cover illustration was produced, and Lisa Marie Sardegna did the postprocessing for the final image. Alan Apt, Mona Pompili, and Sondra Chavez at Prentice Hall tried their best to keep us on schedule and made many helpful suggestions on design and content.

Stuart would like to thank his parents, brother, and sister for their encouragement and their patience at his extended absence. He hopes to be home for Christmas. He would also like to thank Loy Sheflott for her patience and support. He hopes to be home some time tomorrow afternoon. His intellectual debt to his Ph.D. advisor, Michael Genesereth, is evident throughout the book. RUGS (Russell's Unusual Group of Students) have been unusually helpful.

Peter would like to thank his parents (Torsten and Gerda) for getting him started, his advisor (Bob Wilensky), supervisors (Bill Woods and Bob Sproull) and employer (Sun Microsystems) for supporting his work in AI, and his wife (Kris) and friends for encouraging and tolerating him through the long hours of writing.

Before publication, drafts of this book were used in 26 courses by about 1000 students. Both of us deeply appreciate the many comments of these students and instructors (and other reviewers). We can't thank them all individually, but we would like to acknowledge the especially helpful comments of these people:

> Tony Barrett, Howard Beck, John Binder, Larry Bookman, Chris Brown, Lauren Burka, Murray Campbell, Anil Chakravarthy, Roberto Cipolla, Doug Edwards, Kutluhan Erol, Jeffrey Forbes, John Fosler, Bob Futrelle, Sabine Glesner, Barbara Grosz, Steve Hanks, Othar Hansson, Jim Hendler, Tim Huang, Seth Hutchinson, Dan Jurafsky, Leslie Pack Kaelbling, Keiji Kanazawa, Surekha Kasibhatla, Simon Kasif, Daphne Koller, Rich Korf, James Kurien, John Lazzaro, Jason Leatherman, Jon LeBlanc, Jim Martin, Andy Mayer, Steve Minton, Leora Morgenstern, Ron Musick, Stuart Nelson, Steve Omohundro, Ron Parr, Tony Passera, Michael Pazzani, Ira Pohl, Martha Pollack, Bruce Porter, Malcolm Pradhan, Lorraine Prior, Greg Provan, Philip Resnik, Richard Scherl, Daniel Sleator, Robert Sproull, Lynn Stein, Devika Subramanian, Rich Sutton, Jonathan Tash, Austin Tate, Mark Torrance, Randall Upham, Jim Waldo, Bonnie Webber, Michael Wellman, Dan Weld, Richard Yen, Shlomo Zilberstein.

Summary of Contents

I Artificial Intelligence 1

1 Introduction ... 3

2 Intelligent Agents ... 31

II Problem-solving 53

3 Solving Problems by Searching ... 55

4 Informed Search Methods ... 92

5 Game Playing ... 122

III Knowledge and reasoning 149

6 Agents that Reason Logically ... 151

7 First-Order Logic ... 185

8 Building a Knowledge Base ... 217

9 Inference in First-Order Logic ... 265

10 Logical Reasoning Systems ... 297

IV Acting logically 335

11 Planning ... 337

12 Practical Planning ... 367

13 Planning and Acting ... 392

V Uncertain knowledge and reasoning 413

14 Uncertainty ... 415

15 Probabilistic Reasoning Systems ... 436

16 Making Simple Decisions ... 471

17 Making Complex Decisions ... 498

VI Learning 523

18 Learning from Observations ... 525

19 Learning in Neural and Belief Networks ... 563

20 Reinforcement Learning ... 598

21 Knowledge in Learning ... 625

VII Communicating, perceiving, and acting 649

22 Agents that Communicate ... 651

23 Practical Natural Language Processing ... 691

24 Perception ... 724

25 Robotics ... 773

VIII Conclusions 815

26 Philosophical Foundations ... 817

27 AI: Present and Future ... 842

A Complexity analysis and O() notation ... 851

B Notes on Languages and Algorithms ... 854

Bibliography 859

Index 905

Contents

I Artificial Intelligence 1

1 Introduction 3
- 1.1 What is AI? 4
  - Acting humanly: The Turing Test approach 5
  - Thinking humanly: The cognitive modelling approach 6
  - Thinking rationally: The laws of thought approach 6
  - Acting rationally: The rational agent approach 7
- 1.2 The Foundations of Artificial Intelligence 8
  - Philosophy (428 B.C.-present) 8
  - Mathematics (c. 800–present) 11
  - Psychology (1879–present) 12
  - Computer engineering (1940–present) 14
  - Linguistics (1957–present) 15
- 1.3 The History of Artificial Intelligence 16
  - The gestation of artificial intelligence (1943–1956) 16
  - Early enthusiasm, great expectations (1952-1969) 17
  - A dose of reality (1966-1974) 20
  - Knowledge-based systems: The key to power? (1969–1979) 22
  - AI becomes an industry (1980-1988) 24
  - The return of neural networks (1986–present) 24
  - Recent events (1987–present) 25
- 1.4 The State of the Art 26
- 1.5 Summary 27
- Bibliographical and Historical Notes 28
- Exercises 28

2 Intelligent Agents 31
- 2.1 Introduction 31
- 2.2 How Agents Should Act 31
  - The ideal mapping from percept sequences to actions 34
  - Autonomy 35
- 2.3 Structure of Intelligent Agents 35
  - Agent programs 37
  - Why not just look up the answers? 38
  - An example 39
  - Simple reflex agents 40
  - Agents that keep track of the world 41
  - Goal-based agents 42
  - Utility-based agents 44
- 2.4 Environments 45

Properties of environments 46
Environment programs 47
2.5 Summary 49
Bibliographical and Historical Notes 50
Exercises 50

II Problem-solving 53

3 Solving Problems by Searching 55
3.1 Problem-Solving Agents 55
3.2 Formulating Problems 57
Knowledge and problem types 58
Well-defined problems and solutions 60
Measuring problem-solving performance 61
Choosing states and actions 61
3.3 Example Problems 63
Toy problems 63
Real-world problems 68
3.4 Searching for Solutions 70
Generating action sequences 70
Data structures for search trees 72
3.5 Search Strategies 73
Breadth-first search 74
Uniform cost search 75
Depth-first search 77
Depth-limited search 78
Iterative deepening search 78
Bidirectional search 80
Comparing search strategies 81
3.6 Avoiding Repeated States 82
3.7 Constraint Satisfaction Search 83
3.8 Summary 85
Bibliographical and Historical Notes 86
Exercises 87

4 Informed Search Methods 92
4.1 Best-First Search 92
Minimize estimated cost to reach a goal: Greedy search 93
Minimizing the total path cost: A* search 96
4.2 Heuristic Functions 101
The effect of heuristic accuracy on performance 102
Inventing heuristic functions 103
Heuristics for constraint satisfaction problems 104
4.3 Memory Bounded Search 106

Iterative deepening A* search (IDA*) . . . 106
SMA* search . . . 107
4.4 Iterative Improvement Algorithms . . . 111
Hill-climbing search . . . 111
Simulated annealing . . . 113
Applications in constraint satisfaction problems . . . 114
4.5 Summary . . . 115
Bibliographical and Historical Notes . . . 115
Exercises . . . 118

5 Game Playing 122
5.1 Introduction: Games as Search Problems . . . 122
5.2 Perfect Decisions in Two-Person Games . . . 123
5.3 Imperfect Decisions . . . 126
Evaluation functions . . . 127
Cutting off search . . . 129
5.4 Alpha-Beta Pruning . . . 129
Effectiveness of alpha-beta pruning . . . 131
5.5 Games That Include an Element of Chance . . . 133
Position evaluation in games with chance nodes . . . 135
Complexity of expectiminimax . . . 135
5.6 State-of-the-Art Game Programs . . . 136
Chess . . . 137
Checkers or Draughts . . . 138
Othello . . . 138
Backgammon . . . 139
Go . . . 139
5.7 Discussion . . . 139
5.8 Summary . . . 141
Bibliographical and Historical Notes . . . 141
Exercises . . . 145

III Knowledge and reasoning 149

6 Agents that Reason Logically 151
6.1 A Knowledge-Based Agent . . . 151
6.2 The Wumpus World Environment . . . 153
Specifying the environment . . . 154
Acting and reasoning in the wumpus world . . . 155
6.3 Representation, Reasoning, and Logic . . . 157
Representation . . . 160
Inference . . . 163
Logics . . . 165
6.4 Propositional Logic: A Very Simple Logic . . . 166

Syntax 166
Semantics 168
Validity and inference 169
Models 170
Rules of inference for propositional logic 171
Complexity of propositional inference 173
6.5 An Agent for the Wumpus World 174
The knowledge base 174
Finding the wumpus 175
Translating knowledge into action 176
Problems with the propositional agent 176
6.6 Summary 178
Bibliographical and Historical Notes 178
Exercises 180

7 First-Order Logic 185
7.1 Syntax and Semantics 186
Terms 188
Atomic sentences 189
Complex sentences 189
Quantifiers 189
Equality 193
7.2 Extensions and Notational Variations 194
Higher-order logic 195
Functional and predicate expressions using the A operator 195
The uniqueness quantifier ∃! 196
The uniqueness operator ι 196
Notational variations 196
7.3 Using First-Order Logic 197
The kinship domain 197
Axioms, definitions, and theorems 198
The domain of sets 199
Special notations for sets, lists and arithmetic 200
Asking questions and getting answers 200
7.4 Logical Agents for the Wumpus World 201
7.5 A Simple Reflex Agent 202
Limitations of simple reflex agents 203
7.6 Representing Change in the World 203
Situation calculus 204
Keeping track of location 206
7.7 Deducing Hidden Properties of the World 208
7.8 Preferences Among Actions 210
7.9 Toward a Goal-Based Agent 211
7.10 **Summary** 211

Bibliographical and Historical Notes 212
Exercises 213

8 Building a Knowledge Base 217
8.1 Properties of Good and Bad Knowledge Bases 218
8.2 Knowledge Engineering 221
8.3 The Electronic Circuits Domain 223
Decide what to talk about 223
Decide on a vocabulary 224
Encode general rules 225
Encode the specific instance 225
Pose queries to the inference procedure 226
8.4 General Ontology 226
Representing Categories 229
Measures 231
Composite objects 233
Representing change with events 234
Times, intervals, and actions 238
Objects revisited 240
Substances and objects 241
Mental events and mental objects 243
Knowledge and action 247
8.5 The Grocery Shopping World 247
Complete description of the shopping simulation 248
Organizing knowledge 249
Menu-planning 249
Navigating 252
Gathering 253
Communicating 254
Paying 255
8.6 Summary 256
Bibliographical and Historical Notes 256
Exercises 261

9 Inference in First-Order Logic **265**
9.1 Inference Rules Involving Quantifiers 265
9.2 An Example Proof 266
9.3 Generalized Modus Ponens 269
Canonical form 270
Unification 270
Sample proof revisited 271
9.4 Forward and Backward Chaining 272
Forward-chaining algorithm 273
Backward-chaining algorithm 275

9.5 Completeness 276
9.6 Resolution: A Complete Inference Procedure 277
The resolution inference rule 278
Canonical forms for resolution 278
Resolution proofs 279
Conversion to Normal Form 281
Example proof 282
Dealing with equality 284
Resolution strategies 284
9.7 Completeness of resolution 286
9.8 Summary 290
Bibliographical and Historical Notes 291
Exercises 294

10 Logical Reasoning Systems **297**
10.1 Introduction 297
10.2 Indexing, Retrieval, and Unification 299
Implementing sentences and terms 299
Store and fetch 299
Table-based indexing 300
Tree-based indexing 301
The unification algorithm 302
10.3 Logic Programming Systems 304
The Prolog language 304
Implementation 305
Compilation of logic programs 306
Other logic programming languages 308
Advanced control facilities 308
10.4 Theorem Provers 310
Design of a theorem prover 310
Extending Prolog 311
Theorem provers as assistants 312
Practical uses of theorem provers 313
10.5 Forward-Chaining Production Systems 313
Match phase 314
Conflict resolution phase 315
Practical uses of production systems 316
10.6 Frame Systems and Semantic Networks 316
Syntax and semantics of semantic networks 317
Inheritance with exceptions 319
Multiple inheritance 320
Inheritance and change 320
Implementation of semantic networks 321
Expressiveness of semantic networks 323

10.7 Description Logics 323
Practical uses of description logics 325
10.8 Managing Retractions, Assumptions, and Explanations 325
10.9 Summary 327
Bibliographical and Historical Notes 328
Exercises 332

IV Acting logically 335

11 Planning 337
11.1 A Simple Planning Agent 337
11.2 From Problem Solving to Planning 338
11.3 Planning in Situation Calculus 341
11.4 Basic Representations for Planning 343
Representations for states and goals 343
Representations for actions 344
Situation space and plan space 345
Representations for plans 346
Solutions 349
11.5 A Partial-Order Planning Example 349
11.6 A Partial-Order Planning Algorithm 355
11.7 Planning with Partially Instantiated Operators 357
11.8 Knowledge Engineering for Planning 359
The blocks world 359
Shakey's world 360
11.9 Summary 362
Bibliographical and Historical Notes 363
Exercises 364

12 Practical Planning 367
12.1 Practical Planners 367
Spacecraft assembly, integration, and verification 367
Job shop scheduling 369
Scheduling for space missions 369
Buildings, aircraft carriers, and beer factories 371
12.2 Hierarchical Decomposition 371
Extending the language 372
Modifying the planner 374
12.3 Analysis of Hierarchical Decomposition 375
Decomposition and sharing 379
Decomposition versus approximation 380
12.4 More Expressive Operator Descriptions 381
Conditional effects 381
Negated and disjunctive goals 382

Universal quantification 383
A planner for expressive operator descriptions 384
12.5 Resource Constraints 386
Using measures in planning 386
Temporal constraints 388
12.6 Summary 388
Bibliographical and Historical Notes 389
Exercises 390

13 Planning and Acting 392
13.1 Conditional Planning 393
The nature of conditional plans 393
An algorithm for generating conditional plans 395
Extending the plan language 398
13.2 A Simple Replanning Agent 401
Simple replanning with execution monitoring 402
13.3 Fully Integrated Planning and Execution 403
13.4 Discussion and Extensions 407
Comparing conditional planning and replanning 407
Coercion and abstraction 409
13.5 Summary 410
Bibliographical and Historical Notes 411
Exercises 412

V Uncertain knowledge and reasoning 413

14 Uncertainty 415
14.1 Acting under Uncertainty 415
Handling uncertain knowledge 416
Uncertainty and rational decisions 418
Design for a decision-theoretic agent 419
14.2 Basic Probability Notation 420
Prior probability 420
Conditional probability 421
14.3 The Axioms of Probability 422
Why the axioms of probability are reasonable 423
The joint probability distribution 425
14.4 Bayes' Rule and Its Use 426
Applying Bayes' rule: The simple case 426
Normalization 427
Using Bayes' rule: Combining evidence 428
14.5 Where Do Probabilities Come From? 430
14.6 Summary 431
Bibliographical and Historical Notes 431

Exercises 433

15 Probabilistic Reasoning Systems 436
15.1 Representing Knowledge in an Uncertain Domain 436
15.2 The Semantics of Belief Networks 438
Representing the joint probability distribution 439
Conditional independence relations in belief networks 444
15.3 Inference in Belief Networks 445
The nature of probabilistic inferences 446
An algorithm for answering queries 447
15.4 Inference in Multiply Connected Belief Networks 453
Clustering methods 453
Cutset conditioning methods 454
Stochastic simulation methods 455
15.5 Knowledge Engineering for Uncertain Reasoning 456
Case study: The Pathfinder system 457
15.6 Other Approaches to Uncertain Reasoning 458
Default reasoning 459
Rule-based methods for uncertain reasoning 460
Representing ignorance: Dempster-Shafer theory 462
Representing vagueness: Fuzzy sets and fuzzy logic 463
15.7 Summary 464
Bibliographical and Historical Notes 464
Exercises 467

16 Making Simple Decisions 471
16.1 Combining Beliefs and Desires Under Uncertainty 471
16.2 The Basis of Utility Theory 473
Constraints on rational preferences 473
. . . and then there was Utility 474
16.3 Utility Functions 475
The utility of money 476
Utility scales and utility assessment 478
16.4 Multiattribute utility functions 480
Dominance 481
Preference structure and multiattribute utility 483
16.5 Decision Networks 484
Representing a decision problem using decision networks 484
Evaluating decision networks 486
16.6 The Value of Information 487
A simple example 487
A general formula 488
Properties of the value of information 489
Implementing an information-gathering agent 490

16.7 Decision-Theoretic Expert Systems . . . 491
16.8 Summary . . . 493
Bibliographical and Historical Notes . . . 493
Exercises . . . 495

17 Making Complex Decisions **498**
17.1 Sequential Decision Problems . . . 498
17.2 Value Iteration . . . 502
17.3 Policy Iteration . . . 505
17.4 Decision-Theoretic Agent Design . . . 508
The decision cycle of a rational agent . . . 508
Sensing in uncertain worlds . . . 510
17.5 Dynamic Belief Networks . . . 514
17.6 Dynamic Decision Networks . . . 516
Discussion . . . 518
17.7 Summary . . . 519
Bibliographical and Historical Notes . . . 520
Exercises . . . 521

VI Learning **523**

18 Learning from Observations 525
18.1 A General Model of Learning Agents . . . 525
Components of the performance element . . . 527
Representation of the components . . . 528
Available feedback . . . 528
Prior knowledge . . . 528
Bringing it all together . . . 529
18.2 Inductive Learning . . . 529
18.3 Learning Decision Trees . . . 531
Decision trees as performance elements . . . 531
Expressiveness of decision trees . . . 532
Inducing decision trees from examples . . . 534
Assessing the performance of the learning algorithm . . . 538
Practical uses of decision tree learning . . . 538
18.4 Using Information Theory . . . 540
Noise and overfitting . . . 542
Broadening the applicability of decision trees . . . 543
18.5 Learning General Logical Descriptions . . . 544
Hypotheses . . . 544
Examples . . . 545
Current-best-hypothesis search . . . 546
Least-commitment search . . . 549
Discussion . . . 552

18.6 Why Learning Works: Computational Learning Theory 552
  How many examples are needed? 553
  Learning decision lists 555
  Discussion 557
18.7 Summary 558
Bibliographical and Historical Notes 559
Exercises 560

19 Learning in Neural and Belief Networks **563**
19.1 How the Brain Works 564
  Comparing brains with digital computers 565
19.2 Neural Networks 567
  Notation 567
  Simple computing elements 567
  Network structures 570
  Optimal network structure 572
19.3 Perceptrons 573
  What perceptrons can represent 573
  Learning linearly separable functions 575
19.4 Multilayer Feed-Forward Networks 578
  Back-propagation learning 578
  Back-propagation as gradient descent search 580
  Discussion 583
19.5 Applications of Neural Networks 584
  Pronunciation 585
  Handwritten character recognition 586
  Driving 586
19.6 Bayesian Methods for Learning Belief Networks 588
  Bayesian learning 588
  Belief network learning problems 589
  Learning networks with fixed structure 589
  A comparison of belief networks and neural networks 592
19.7 Summary 593
Bibliographical and Historical Notes 594
Exercises 596

20 Reinforcement Learning **598**
20.1 Introduction 598
20.2 Passive Learning in a Known Environment 600
  Naïve updating 601
  Adaptive dynamic programming 603
  Temporal difference learning 604
20.3 Passive Learning in an Unknown Environment 605
20.4 Active Learning in an Unknown Environment 607

20.5 Exploration 609
20.6 Learning an Action-Value Function 612
20.7 Generalization in Reinforcement Learning 615
Applications to game-playing 617
Application to robot control 617
20.8 Genetic Algorithms and Evolutionary Programming 619
20.9 Summary 621
Bibliographical and Historical Notes 622
Exercises 623

21 Knowledge in Learning **625**
21.1 Knowledge in Learning 625
Some simple examples 626
Some general schemes 627
21.2 Explanation-Based Learning 629
Extracting general rules from examples 630
Improving efficiency 631
21.3 Learning Using Relevance Information 633
Determining the hypothesis space 633
Learning and using relevance information 634
21.4 Inductive Logic Programming 636
An example 637
Inverse resolution 639
Top-down learning methods 641
21.5 Summary 644
Bibliographical and Historical Notes 645
Exercises 647

VII Communicating, perceiving, and acting **649**

22 **Agents that Communicate** **651**
22.1 Communication as Action 652
Fundamentals of language 654
The component steps of communication 655
Two models of communication 659
22.2 Types of Communicating Agents 659
Communicating using Tell and Ask 660
Communicating using formal language 661
An agent that communicates 662
22.3 A Formal Grammar for a Subset of English 662
The Lexicon of $\mathcal{E}_0$ 664
The Grammar of $\mathcal{E}_0$ 664
22.4 Syntactic Analysis (Parsing) 664
22.5 Definite Clause Grammar (DCG) 667

22.6 Augmenting a Grammar . . . 668
Verb Subcategorization . . . 669
Generative Capacity of Augmented Grammars . . . 671
22.7 Semantic Interpretation . . . 672
Semantics as DCG Augmentations . . . 673
The semantics of "John loves Mary" . . . 673
The semantics of $\mathcal{E}_1$ . . . 675
Converting quasi-logical form to logical form . . . 677
Pragmatic Interpretation . . . 678
22.8 Ambiguity and Disambiguation . . . 680
Disambiguation . . . 682
22.9 A Communicating Agent . . . 683
22.10 Summary . . . 684
Bibliographical and Historical Notes . . . 685
Exercises . . . 688

23 Practical Natural Language Processing 691
23.1 Practical Applications . . . 691
Machine translation . . . 691
Database access . . . 693
Information retrieval . . . 694
Text categorization . . . 695
Extracting data from text . . . 696
23.2 Efficient Parsing . . . 696
Extracting parses from the chart: Packing . . . 701
23.3 Scaling Up the Lexicon . . . 703
23.4 Scaling Up the Grammar . . . 705
Nominal compounds and apposition . . . 706
Adjective phrases . . . 707
Determiners . . . 708
Noun phrases revisited . . . 709
Clausal complements . . . 710
Relative clauses . . . 710
Questions . . . 711
Handling agrammatical strings . . . 712
23.5 Ambiguity . . . 712
Syntactic evidence . . . 713
Lexical evidence . . . 713
Semantic evidence . . . 713
Metonymy . . . 714
Metaphor . . . 715
23.6 Discourse Understanding . . . 715
The structure of coherent discourse . . . 717
23.7 Summary . . . 719

Bibliographical and Historical Notes 720
Exercises 721

24 Perception 724
24.1 Introduction 724
24.2 Image Formation 725
  Pinhole camera 725
  Lens systems 727
  Photometry of image formation 729
  Spectrophotometry of image formation 730
24.3 Image-Processing Operations for Early Vision 730
  Convolution with linear filters 732
  Edge detection 733
24.4 Extracting 3-D Information Using Vision 734
  Motion 735
  Binocular stereopsis 737
  Texture gradients 742
  Shading 743
  Contour 745
24.5 Using Vision for Manipulation and Navigation 749
24.6 Object Representation and Recognition 751
  The alignment method 752
  Using projective invariants 754
24.7 Speech Recognition 757
  Signal processing 758
  Defining the overall speech recognition model 760
  The language model: P(words) 760
  The acoustic model: P(signallwords) 762
  Putting the models together 764
  The search algorithm 765
  Training the model 766
24.8 Summary 767
Bibliographical and Historical Notes 767
Exercises 771

25 **Robotics** 773
25.1 Introduction 773
25.2 Tasks: What Are Robots Good For? 774
  Manufacturing and materials handling 774
  Gofer robots 775
  Hazardous environments 775
  Telepresence and virtual reality 776
  Augmentation of human abilities 776
25.3 Parts: What Are Robots Made Of? 777

Effectors: Tools for action 777
Sensors: Tools for perception 782
25.4 Architectures 786
Classical architecture 787
Situated automata 788
25.5 Configuration Spaces: A Framework for Analysis 790
Generalized configuration space 792
Recognizable Sets 795
25.6 Navigation and Motion Planning 796
Cell decomposition 796
Skeletonization methods 798
Fine-motion planning 802
Landmark-based navigation 805
Online algorithms 806
25.7 Summary 809
Bibliographical and Historical Notes 809
Exercises 811

VIII Conclusions 815

26 Philosophical Foundations 817
26.1 The Big Questions 817
26.2 Foundations of Reasoning and Perception 819
26.3 On the Possibility of Achieving Intelligent Behavior 822
The mathematical objection 824
The argument from informality 826
26.4 Intentionality and Consciousness 830
The Chinese Room 831
The Brain Prosthesis Experiment 835
Discussion 836
26.5 Summary 837
Bibliographical and Historical Notes 838
Exercises 840

27 **AI: Present and Future 842**
27.1 Have We Succeeded Yet? 842
27.2 What Exactly Are We Trying to Do? 845
27.3 What If We Do Succeed? 848

A Complexity analysis and O() notation 851
A.1 Asymptotic Analysis 851
A.2 Inherently Hard Problems 852
Bibliographical and Historical Notes 853

B Notes on Languages and Algorithms **854**
B.1 Defining Languages with Backus-Naur Form (BNF) 854
B.2 Describing Algorithms with Pseudo-Code 855
Nondeterminism 855
Static variables 856
Functions as values 856
B.3 The Code Repository 857
B.4 Comments 857

Bibliography 859

Index **905**

Part I

ARTIFICIAL INTELLIGENCE

The two chapters in this part introduce the subject of Artificial Intelligence or AI and our approach to the subject: that AI is the study of *agents* that exist in an environment and perceive and act.

and subtracting machine called the Pascaline. Leibniz improved on this in 1694, building a mechanical device that multiplied by doing repeated addition. Progress stalled for over a century until Charles Babbage (1792–1871) dreamed that logarithm tables could be computed by machine. He designed a machine for this task, but never completed the project. Instead, he turned to the design of the Analytical Engine, for which Babbage invented the ideas of addressable memory, stored programs, and conditional jumps. Although the idea of programmable machines was not new—in 1805, Joseph Marie Jacquard invented a loom that could be programmed using punched cards—Babbage's machine was the first artifact possessing the characteristics necessary for universal computation. Babbage's colleague Ada Lovelace, daughter of the poet Lord Byron, wrote programs for the Analytical Engine and even speculated that the machine could play chess or compose music. Lovelace was the world's first programmer, and the first of many to endure massive cost overruns and to have an ambitious project ultimately abandoned.[11] Babbage's basic design was proven viable by Doron Swade and his colleagues, who built a working model using only the mechanical techniques available at Babbage's time (Swade, 1993). Babbage had the right idea, but lacked the organizational skills to get his machine built.

AI also owes a debt to the software side of computer science, which has supplied the operating systems, programming languages, and tools needed to write modern programs (and papers about them). But this is one area where the debt has been repaid: work in AI has pioneered many ideas that have made their way back to "mainstream" computer science, including time sharing, interactive interpreters, the linked list data type, automatic storage management, and some of the key concepts of object-oriented programming and integrated program development environments with graphical user interfaces.

Linguistics (1957–present)

In 1957, B. F. Skinner published *Verbal Behavior.* This was a comprehensive, detailed account of the behaviorist approach to language learning, written by the foremost expert in the field. But curiously, a review of the book became as well-known as the book itself, and served to almost kill off interest in behaviorism. The author of the review was Noam Chomsky, who had just published a book on his own theory, *Syntactic Structures.* Chomsky showed how the behaviorist theory did not address the notion of creativity in language—it did not explain how a child could understand and make up sentences that he or she had never heard before. Chomsky's theory—based on syntactic models going back to the Indian linguist Panini (c. 350 B.C.)—could explain this, and unlike previous theories, it was formal enough that it could in principle be programmed.

Later developments in linguistics showed the problem to be considerably more complex than it seemed in 1957. Language is ambiguous and leaves much unsaid. This means that understanding language requires an understanding of the subject matter and context, not just an understanding of the structure of sentences. This may seem obvious, but it was not appreciated until the early 1960s. Much of the early work in **knowledge representation** (the study of how to put knowledge into a form that a computer can reason with) was tied to language and informed by research in linguistics, which was connected in turn to decades of work on the philosophical analysis of language.

---

[11] She also gave her name to Ada, the U.S. Department of Defense's all-purpose programming language.

1 INTRODUCTION

In which we try to explain why we consider artificial intelligence to be a subject most worthy of study, and in which we try to decide what exactly it is, this being a good thing to decide before embarking.

ARTIFICIAL INTELLIGENCE

Humankind has given itself the scientific name **homo sapiens**—man the wise—because our mental capacities are so important to our everyday lives and our sense of self. The field of **artificial intelligence,** or AI, attempts to understand intelligent entities. Thus, one reason to study it is to learn more about ourselves. But unlike philosophy and psychology, which are also concerned with intelligence, AI strives to *build* intelligent entities as well as understand them. Another reason to study AI is that these constructed intelligent entities are interesting and useful in their own right. AI has produced many significant and impressive products even at this early stage in its development. Although no one can predict the future in detail, it is clear that computers with human-level intelligence (or better) would have a huge impact on our everyday lives and on the future course of civilization.

AI addresses one of the ultimate puzzles. How is it possible for a slow, tiny brain, whether biological or electronic, to perceive, understand, predict, and manipulate a world far larger and more complicated than itself? How do we go about making something with those properties? These are hard questions, but unlike the search for faster-than-light travel or an antigravity device, the researcher in AI has solid evidence that the quest is possible. All the researcher has to do is look in the mirror to see an example of an intelligent system.

AI is one of the newest disciplines. It was formally initiated in 1956, when the name was coined, although at that point work had been under way for about five years. Along with modern genetics, it is regularly cited as the "field I would most like to be in" by scientists in other disciplines. A student in physics might reasonably feel that all the good ideas have already been taken by Galileo, Newton, Einstein, and the rest, and that it takes many years of study before one can contribute new ideas. AI, on the other hand, still has openings for a full-time Einstein.

The study of intelligence is also one of the oldest disciplines. For over 2000 years, philosophers have tried to understand how seeing, learning, remembering, and reasoning could, or should, be done.[1] The advent of usable computers in the early 1950s turned the learned but armchair speculation concerning these mental faculties into a real experimental and theoretical discipline. Many felt that the new "Electronic Super-Brains" had unlimited potential for intelligence. "Faster Than Einstein" was a typical headline. But as well as providing a vehicle for creating artificially intelligent entities, the computer provides a tool for testing theories of intelligence, and many theories failed to withstand the test—a case of "out of the armchair, into the fire." AI has turned out to be more difficult than many at first imagined, and modern ideas are much richer, more subtle, and more interesting as a result.

AI currently encompasses a huge variety of subfields, from general-purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. Often, scientists in other fields move gradually into artificial intelligence, where they find the tools and vocabulary to systematize and automate the intellectual tasks on which they have been working all their lives. Similarly, workers in AI can choose to apply their methods to any area of human intellectual endeavor. In this sense, it is truly a universal field.

1.1 WHAT IS AI?

We have now explained why AI is exciting, but we have not said what it *is*. We could just say, "Well, it has to do with smart programs, so let's get on and write some." But the history of science shows that it is helpful to aim at the right goals. Early alchemists, looking for a potion for eternal life and a method to turn lead into gold, were probably off on the wrong foot. Only when the aim changed, to that of finding explicit theories that gave accurate predictions of the terrestrial world, in the same way that early astronomy predicted the apparent motions of the stars and planets, could the scientific method emerge and productive science take place.

RATIONALITY

Definitions of artificial intelligence according to eight recent textbooks are shown in Figure 1.1. These definitions vary along two main dimensions. The ones on top are concerned with *thought processes* and *reasoning,* whereas the ones on the bottom address *behavior.* Also, the definitions on the left measure success in terms of *human* performance, whereas the ones on the right measure against an *ideal* concept of intelligence, which we will call **rationality.** A system is rational if it does the right thing. This gives us four possible goals to pursue in artificial intelligence, as seen in the caption of Figure 1.1.

Historically, all four approaches have been followed. As one might expect, a tension exists between approaches centered around humans and approaches centered around rationality.[2] A human-centered approach must be an empirical science, involving hypothesis and experimental

---

[1] A more recent branch of philosophy is concerned with proving that AI is impossible. We will return to this interesting viewpoint in Chapter 26.

[2] We should point out that by distinguishing between *human* and *rational* behavior, we are not suggesting that humans are necessarily "irrational" in the sense of "emotionally unstable" or "insane." One merely need note that we often make mistakes; we are not all chess grandmasters even though we may know all the rules of chess; and unfortunately, not everyone gets an A on the exam. Some systematic errors in human reasoning are cataloged by Kahneman *et al.* (1982).

"The exciting new effort to make computers think . . . *machines with minds,* in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Figure 1.1 Some definitions of AI. They are organized into four categories:

Systems that think like humans.	Systems that think rationally.
Systems that act like humans.	Systems that act rationally.

confirmation. A rationalist approach involves a combination of mathematics and engineering. People in each group sometimes cast aspersions on work done in the other groups, but the truth is that each direction has yielded valuable insights. Let us look at each in more detail.

Acting humanly: The Turing Test approach

TURING TEST

The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence. Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator. Roughly speaking, the test he proposed is that the computer should be interrogated by a human via a teletype, and passes the test if the interrogator cannot tell if there is a computer or a human at the other end. Chapter 26 discusses the details of the test, and whether or not a computer is really intelligent if it passes. For now, programming a computer to pass the test provides plenty to work on. The computer would need to possess the following capabilities:

NATURAL LANGUAGE PROCESSING

0 **natural language processing** to enable it to communicate successfully in English (or some other human language);

KNOWLEDGE REPRESENTATION

◇ **knowledge representation** to store information provided before or during the interrogation;

AUTOMATED REASONING

◇ **automated reasoning** to use the stored information to answer questions and to draw new conclusions;

MACHINE LEARNING

◇ **machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because *physical* simulation of a person is unnecessary for intelligence. However,

TOTAL TURING TEST

the so-called **total Turing Test** includes a video signal so that the interrogator can test the subject's perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch." To pass the total Turing Test, the computer will need

COMPUTER VISION

ROBOTICS

- ◊ **computer vision** to perceive objects, and
- ◊ **robotics** to move them about.

Within AI, there has not been a big effort to try to pass the Turing test. The issue of acting like a human comes up primarily when AI programs have to interact with people, as when an expert system explains how it came to its diagnosis, or a natural language processing system has a dialogue with a user. These programs must behave according to certain normal conventions of human interaction in order to make themselves understood. The underlying representation and reasoning in such a system may or may not be based on a human model.

Thinking humanly: The cognitive modelling approach

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get *inside* the actual workings of human minds. There are two ways to do this: through introspection—trying to catch our own thoughts as they go by—or through psychological experiments. Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program's input/output and timing behavior matches human behavior, that is evidence that some of the program's mechanisms may also be operating in humans. For example, Newell and Simon, who developed GPS, the "General Problem Solver" (Newell and Simon, 1961), were not content to have their program correctly solve problems. They were more concerned with comparing the trace of its reasoning steps to traces of human subjects solving the same problems. This is in contrast to other researchers of the same time (such as Wang (1960)), who were concerned with getting the right answers regardless of how humans might do it. The interdisciplinary field of **cognitive science** brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind.

COGNITIVE SCIENCE

Although cognitive science is a fascinating field in itself, we are not going to be discussing it all that much in this book. We will occasionally comment on similarities or differences between AI techniques and human cognition. Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals, and we assume that the reader only has access to a computer for experimentation. We will simply note that AI and cognitive science continue to fertilize each other, especially in the areas of vision, natural language, and learning. The history of psychological theories of cognition is briefly covered on page 12.

Thinking rationally: The laws of thought approach

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes. His famous **syllogisms** provided patterns for argument structures that always gave correct conclusions given correct premises. For example, "Socrates is a man;

SYLLOGISMS

LOGIC

all men are mortal; therefore Socrates is mortal." These laws of thought were supposed to govern the operation of the mind, and initiated the field of **logic.**

LOGICIST

The development of formal logic in the late nineteenth and early twentieth centuries, which we describe in more detail in Chapter 6, provided a precise notation for statements about all kinds of things in the world and the relations between them. (Contrast this with ordinary arithmetic notation, which provides mainly for equality and inequality statements about numbers.) By 1965, programs existed that could, given enough time and memory, take a description of a problem in logical notation and find the solution to the problem, if one exists. (If there is no solution, the program might never stop looking for it.) The so-called **logicist** tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between being able to solve a problem "in principle" and doing so in practice. Even problems with just a few dozen facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first. Although both of these obstacles apply to *any* attempt to build computational reasoning systems, they appeared first in the logicist tradition because the power of the representation and reasoning systems are well-defined and fairly well understood.

Acting rationally: The rational agent approach

AGENT

Acting rationally means acting so as to achieve one's goals, given one's beliefs. An **agent** is just something that perceives and acts. (This may be an unusual use of the word, but you will get used to it.) In this approach, AI is viewed as the study and construction of rational agents.

In the "laws of thought" approach to AI, the whole emphasis was on correct inferences. Making correct inferences is sometimes *part* of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals, and then to act on that conclusion. On the other hand, correct inference is not *all* of rationality, because there are often situations where there is no provably correct thing to do, yet something must still be done. There are also ways of acting rationally that cannot be reasonably said to involve inference. For example, pulling one's hand off of a hot stove is a reflex action that is more successful than a slower action taken after careful deliberation.

All the "cognitive skills" needed for the Turing Test are there to allow rational actions. Thus, we need the ability to represent knowledge and reason with it because this enables us to reach good decisions in a wide variety of situations. We need to be able to generate comprehensible sentences in natural language because saying those sentences helps us get by in a complex society. We need learning not just for erudition, but because having a better idea of how the world works enables us to generate more effective strategies for dealing with it. We need visual perception not just because seeing is fun, but in order to get a better idea of what an action might achieve—for example, being able to see a tasty morsel helps one to move toward it.

The study of AI as rational agent design therefore has two advantages. First, it is more general than the "laws of thought" approach, because correct inference is only a useful mechanism for achieving rationality, and not a necessary one. Second, it is more amenable to scientific development than approaches based on human behavior or human thought, because the standard of rationality is clearly defined and completely general. Human behavior, on the other hand, is well-adapted for one specific environment and is the product, in part, of a complicated and largely unknown evolutionary process that still may be far from achieving perfection. *This book will therefore concentrate on general principles of rational agents, and on components for constructing them.* We will see that despite the apparent simplicity with which the problem can be stated, an enormous variety of issues come up when we try to solve it. Chapter 2 outlines some of these issues in more detail.

One important point to keep in mind: we will see before too long that achieving perfect rationality—always doing the right thing—is not possible in complicated environments. The computational demands are just too high. However, for most of the book, we will adopt the working hypothesis that understanding perfect decision making is a good place to start. It simplifies the problem and provides the appropriate setting for most of the foundational material in the field. Chapters 5 and 17 deal explicitly with the issue of **limited rationality**—acting appropriately when there is not enough time to do all the computations one might like.

LIMITED RATIONALITY

1.2 THE FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

In this section and the next, we provide a brief history of AI. Although AI itself is a young field, it has inherited many ideas, viewpoints, and techniques from other disciplines. From over 2000 years of tradition in philosophy, theories of reasoning and learning have emerged, along with the viewpoint that the mind is constituted by the operation of a physical system. From over 400 years of mathematics, we have formal theories of logic, probability, decision making, and computation. From psychology, we have the tools with which to investigate the human mind, and a scientific language within which to express the resulting theories. From linguistics, we have theories of the structure and meaning of language. Finally, from computer science, we have the tools with which to make AI a reality.

Like any history, this one is forced to concentrate on a small number of people and events, and ignore others that were also important. We choose to arrange events to tell the story of how the various intellectual components of modern AI came into being. We certainly would not wish to give the impression, however, that the disciplines from which the components came have all been working toward AI as their ultimate fruition.

Philosophy (428 B.C.-present)

> The safest characterization of the European philosophical tradition is that it consists of a series of footnotes to Plato.
> —Alfred North Whitehead

We begin with the birth of Plato in 428 B.C. His writings range across politics, mathematics, physics, astronomy, and several branches of philosophy. Together, Plato, his teacher Socrates, and his student Aristotle laid the foundation for much of western thought and culture. The philosopher Hubert Dreyfus (1979, p. 67) says that "The story of artificial intelligence might well begin around 450 B.C." when Plato reported a dialogue in which Socrates asks Euthyphro,[3] "I want to know what is characteristic of piety which makes all actions pious . . . that I may have it to turn to, and to use as a standard whereby to judge your actions and those of other men."[4] In other words, Socrates was asking for an *algorithm* to distinguish piety from non-piety. Aristotle went on to try to formulate more precisely the laws governing the rational part of the mind. He developed an informal system of syllogisms for proper reasoning, which in principle allowed one to mechanically generate conclusions, given initial premises. Aristotle did not believe all parts of the mind were governed by logical processes; he also had a notion of intuitive reason.

Now that we have the idea of a set of rules that can describe the working of (at least part of) the mind, the next step is to consider the mind as a physical system. We have to wait for René Descartes (1596–1650) for a clear discussion of the distinction between mind and matter, and the problems that arise. One problem with a purely physical conception of the mind is that it seems to leave little room for free will: if the mind is governed entirely by physical laws, then it has no more free will than a rock "deciding" to fall toward the center of the earth. Although a strong advocate of the power of reasoning, Descartes was also a proponent **of dualism.** He held that there is a part of the mind (or soul or spirit) that is outside of nature, exempt from physical laws. On the other hand, he felt that animals did not possess this dualist quality; they could be considered as if they were machines.

DUALISM

MATERIALISM

An alternative to dualism is **materialism,** which holds that all the world (including the brain and mind) operate according to physical law.[5] Wilhelm Leibniz (1646–1716) was probably the first to take the materialist position to its logical conclusion and build a mechanical device intended to carry out mental operations. Unfortunately, his formulation of logic was so weak that his mechanical concept generator could not produce interesting results.

It is also possible to adopt an intermediate position, in which one accepts that the mind has a physical basis, but denies that it can be *explained* by a reduction to ordinary physical processes. Mental processes and consciousness are therefore part of the physical world, but inherently unknowable; they are beyond rational understanding. Some philosophers critical of AI have adopted exactly this position, as we discuss in Chapter 26.

EMPIRICIST

INDUCTION

Barring these possible objections to the aims of AI, philosophy had thus established a tradition in which the mind was conceived of as a physical device operating principally by reasoning with the knowledge that it contained. The next problem is then to establish the source of knowledge. The **empiricist** movement, starting with Francis Bacon's (1561-1626) *Novum Organum*,[6] is characterized by the dictum of John Locke (1632–1704): "Nothing is in the understanding, which was not first in the senses." David Hume's (1711–1776) *A Treatise of Human Nature* (Hume, 1978) proposed what is now known as the principle of **induction:**

---

[3] The *Euthyphro* describes the events just before the trial of Socrates in 399 B.C. Dreyfus has clearly erred in placing it 51 years earlier.

[4] Note that other translations have "goodness/good" instead of "piety/pious."

[5] In this view, the perception of "free will" arises because the deterministic generation of behavior is constituted by the operation of the mind selecting among what appear to be the possible courses of action. They remain "possible" because the brain does not have access to its own future states.

[6] An update of Aristotle's *organon*, or instrument of thought.

that general rules are acquired by exposure to repeated associations between their elements. The theory was given more formal shape by Bertrand Russell (1872-1970) who introduced **logical positivism.** This doctrine holds that all knowledge can be characterized by logical theories connected, ultimately, to **observation sentences** that correspond to sensory inputs.[7] The **confirmation theory** of Rudolf Carnap and Carl Hempel attempted to establish the nature of the connection between the observation sentences and the more general theories—in other words, to understand how knowledge can be acquired from experience.

LOGICAL POSITIVISM

OBSERVATION SENTENCES

CONFIRMATION THEORY

The final element in the philosophical picture of the mind is the connection between knowledge and action. What form should this connection take, and how can particular actions be justified? These questions are vital to AI, because only by understanding how actions are justified can we understand how to build an agent whose actions are justifiable, or rational. Aristotle provides an elegant answer in the *Nicomachean Ethics* (Book III. 3, 1112b):

> We deliberate not about ends, but about means. For a doctor does not deliberate whether he shall heal, nor an orator whether he shall persuade, nor a statesman whether he shall produce law and order, nor does any one else deliberate about his end. They assume the end and consider how and by what means it is attained, and if it seems easily and best produced thereby; while if it is achieved by one means only they consider *how* it will be achieved by this and by what means *this* will be achieved, till they come to the first cause, which in the order of discovery is last . . . and what is last in the order of analysis seems to be first in the order of becoming. And if we come on an impossibility, we give up the search, e.g. if we need money and this cannot be got: but if a thing appears possible we try to do it.

Aristotle's approach (with a few minor refinements) was implemented 2300 years later by Newell and Simon in their GPS program, about which they write (Newell and Simon, 1972):

MEANS–ENDS ANALYSIS

> The main methods of GPS jointly embody the heuristic **of means-ends analysis.** Means–ends analysis is typified by the following kind of common-sense argument:
>
> > I want to take my son to nursery school. What's the difference between what I have and what I want? One of distance. What changes distance? My automobile. My automobile won't work. What is needed to make it work? A new battery. What has new batteries? An auto repair shop. I want the repair shop to put in a new battery; but the shop doesn't know I need one. What is the difficulty? One of communication. What allows communication? A telephone . . . and so on.
>
> This kind of analysis—classifying things in terms of the functions they serve and oscillating among ends, functions required, and means that perform them—forms the basic system of heuristic of GPS.

Means-ends analysis is useful, but does not say what to do when several actions will achieve the goal, or when no action will completely achieve it. Arnauld, a follower of Descartes, correctly described a quantitative formula for deciding what action to take in cases like this (see Chapter 16). John Stuart Mill's (1806–1873) book *Utilitarianism* (Mill, 1863) amplifies on this idea. The more formal theory of decisions is discussed in the following section.

---

[7] In this picture, all meaningful statements can be verified or falsified either by analyzing the meaning of the words or by carrying out experiments. Because this rules out most of metaphysics, as was the intention, logical positivism was unpopular in some circles.

Mathematics (c. 800-present)

Philosophers staked out most of the important ideas of AI, but to make the leap to a formal science required a level of mathematical formalization in three main areas: computation, logic, and probability. The notion of expressing a computation as a formal **algorithm** goes back to al-Khowarazmi, an Arab mathematician of the ninth century, whose writings also introduced Europe to Arabic numerals and algebra.

ALGORITHM

Logic goes back at least to Aristotle, but it was a philosophical rather than mathematical subject until George Boole (1815-1864) introduced his formal language for making logical inference in 1847. Boole's approach was incomplete, but good enough that others filled in the gaps. In 1879, Gottlob Frege (1848-1925) produced a logic that, except for some notational changes, forms the first-order logic that is used today as the most basic knowledge representation system.[8] Alfred Tarski (1902-1983) introduced a theory of reference that shows how to relate the objects in a logic to objects in the real world. The next step was to determine the limits of what could be done with logic and computation.

David Hilbert (1862-1943), a great mathematician in his own right, is most remembered for the problems he did not solve. In 1900, he presented a list of 23 problems that he correctly predicted would occupy mathematicians for the bulk of the century. The final problem asks if there is an algorithm for deciding the truth of any logical proposition involving the natural numbers—the famous *Entscheidungsproblem*, or decision problem. Essentially, Hilbert was asking if there were fundamental limits to the power of effective proof procedures. In 1930, Kurt Godel (1906-1978) showed that there exists an effective procedure to prove any true statement in the first-order logic of Frege and Russell; but first-order logic could not capture the principle of mathematical induction needed to characterize the natural numbers. In 1931, he showed that real limits do exist. His **incompleteness theorem** showed that in any language expressive enough to describe the properties of the natural numbers, there are true statements that are undecidable: their truth cannot be established by any algorithm.

INCOMPLETENESS THEOREM

This fundamental result can also be interpreted as showing that there are some functions on the integers that cannot be represented by an algorithm—that is, they cannot be computed. This motivated Alan Turing (1912-1954) to try to characterize exactly which functions *are* capable of being computed. This notion is actually slightly problematic, because the notion of a computation or effective procedure really cannot be given a formal definition. However, the Church-Turing thesis, which states that the Turing machine (Turing, 1936) is capable of computing any computable function, is generally accepted as providing a sufficient definition. Turing also showed that there were some functions that no Turing machine can compute. For example, no machine can tell *in general* whether a given program will return an answer on a given input, or run forever.

Although undecidability and noncomputability are important to an understanding of computation, the notion of **intractability** has had a much greater impact. Roughly speaking, a class of problems is called intractable if the time required to solve instances of the class grows at least exponentially with the size of the instances. The distinction between polynomial and exponential growth in complexity was first emphasized in the mid-1960s (Cobham, 1964; Edmonds, 1965). It is important because exponential growth means that even moderate-sized in-

INTRACTABILITY

---

[8] To understand why Frege's notation was not universally adopted, see the cover of this book.

REDUCTION

stances cannot be solved in any reasonable time. Therefore, one should strive to divide the overall problem of generating intelligent behavior into tractable subproblems rather than intractable ones. The second important concept in the theory of complexity is **reduction,** which also emerged in the 1960s (Dantzig, 1960; Edmonds, 1962). A reduction is a general transformation from one class of problems to another, such that solutions to the first class can be found by reducing them to problems of the second class and solving the latter problems.

NP COMPLETENESS

How can one recognize an intractable problem? The theory of **NP-completeness**, pioneered by Steven Cook (1971) and Richard Karp (1972), provides a method. Cook and Karp showed the existence of large classes of canonical combinatorial search and reasoning problems that are NP-complete. Any problem class to which an NP-complete problem class can be reduced is likely to be intractable. (Although it has not yet been proved that NP-complete problems are necessarily intractable, few theoreticians believe otherwise.) These results contrast sharply with the "Electronic Super-Brain" enthusiasm accompanying the advent of computers. Despite the ever-increasing speed of computers, subtlety and careful use of resources will characterize intelligent systems. Put crudely, the world is an *extremely* large problem instance!

Besides logic and computation, the third great contribution of mathematics to AI is the theory of probability. The Italian Gerolamo Cardano (1501-1576) first framed the idea of probability, describing it in terms of the possible outcomes of gambling events. Before his time, the outcomes of gambling games were seen as the will of the gods rather than the whim of chance. Probability quickly became an invaluable part of all the quantitative sciences, helping to deal with uncertain measurements and incomplete theories. Pierre Fermat (1601–1665), Blaise Pascal (1623-1662), James Bernoulli (1654-1705), Pierre Laplace (1749-1827), and others advanced the theory and introduced new statistical methods. Bernoulli also framed an alternative view of probability, as a subjective "degree of belief" rather than an objective ratio of outcomes. Subjective probabilities therefore can be updated as new evidence is obtained. Thomas Bayes (1702-1761) proposed a rule for updating subjective probabilities in the light of new evidence (published posthumously in 1763). Bayes' rule, and the subsequent field of Bayesian analysis, form the basis of the modern approach to uncertain reasoning in AI systems. Debate still rages between supporters of the objective and subjective views of probability, but it is not clear if the difference has great significance for AI. Both versions obey the same set of axioms. Savage's (1954) *Foundations of Statistics* gives a good introduction to the field.

DECISION THEORY

As with logic, a connection must be made between probabilistic reasoning and action. **Decision theory,** pioneered by John Von Neumann and Oskar Morgenstern (1944), combines probability theory with utility theory (which provides a formal and complete framework for specifying the preferences of an agent) to give the first general theory that can distinguish good actions from bad ones. Decision theory is the mathematical successor to utilitarianism, and provides the theoretical basis for many of the agent designs in this book.

Psychology (1879–present)

Scientific psychology can be said to have begun with the work of the German physicist Hermann von Helmholtz (1821-1894) and his student Wilhelm Wundt (1832-1920). Helmholtz applied the scientific method to the study of human vision, and his *Handbook of Physiological Optics* is even now described as "the single most important treatise on the physics and physiology of human vision to this day" (Nalwa, 1993, p.15). In 1879, the same year that Frege launched first-order logic, Wundt opened the first laboratory of experimental psychology at the University of Leipzig. Wundt insisted on carefully controlled experiments in which his workers would perform a perceptual or associative task while introspecting on their thought processes. The careful controls went a long way to make psychology a science, but as the methodology spread, a curious phenomenon arose: each laboratory would report introspective data that just happened to match the theories that were popular in that laboratory. The **behaviorism** movement of John Watson (1878–1958) and Edward Lee Thorndike (1874–1949) rebelled against this subjectivism, rejecting *any* theory involving mental processes on the grounds that introspection could not provide reliable evidence. Behaviorists insisted on studying only objective measures of the percepts (or *stimulus)* given to an animal and its resulting actions (or *response).* Mental constructs such as knowledge, beliefs, goals, and reasoning steps were dismissed as unscientific "folk psychology." Behaviorism discovered a lot about rats and pigeons, but had less success understanding humans. Nevertheless, it had a stronghold on psychology (especially in the United States) from about 1920 to 1960.

BEHAVIORISM

COGNITIVE PSYCHOLOGY

The view that the brain possesses and processes information, which is the principal characteristic of **cognitive psychology,** can be traced back at least to the works of William James[9] (1842–1910). Helmholtz also insisted that perception involved a form of unconscious logical inference. The cognitive viewpoint was largely eclipsed by behaviorism until 1943, when Kenneth Craik published *The Nature of Explanation.* Craik put back the missing mental step between stimulus and response. He claimed that beliefs, goals, and reasoning steps could be useful valid components of a theory of human behavior, and are just as scientific as, say, using pressure and temperature to talk about gases, despite their being made of molecules that have neither. Craik specified the three key steps of a knowledge-based agent: (1) the stimulus must be translated into an internal representation, (2) the representation is manipulated by cognitive processes to derive new internal representations, and (3) these are in turn retranslated back into action. He clearly explained why this was a good design for an agent:

> If the organism carries a "small-scale model" of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and future, and in every way to react in a much fuller, safer, and more competent manner to the emergencies which face it. (Craik, 1943)

An agent designed this way can, for example, plan a long trip by considering various possible routes, comparing them, and choosing the best one, all before starting the journey. Since the 1960s, the information-processing view has dominated psychology. It it now almost taken for granted among many psychologists that "a cognitive theory should be like a computer program" (Anderson, 1980). By this it is meant that the theory should describe cognition as consisting of well-defined transformation processes operating at the level of the information carried by the input signals.

For most of the early history of AI and cognitive science, no significant distinction was drawn between the two fields, and it was common to see AI programs described as psychological results without any claim as to the exact human behavior they were modelling. In the last decade or so, however, the methodological distinctions have become clearer, and most work now falls into one field or the other.

---

[9] William James was the brother of novelist Henry James. It is said that Henry wrote fiction as if it were psychology and William wrote psychology as if it were fiction.

Computer engineering (1940–present)

For artificial intelligence to succeed, we need two things: intelligence and an artifact. The computer has been unanimously acclaimed as the artifact with the best chance of demonstrating intelligence. The modern digital electronic computer was invented independently and almost simultaneously by scientists in three countries embattled in World War II. The first operational modern computer was the Heath Robinson,[10] built in 1940 by Alan Turing's team for the single purpose of deciphering German messages. When the Germans switched to a more sophisticated code, the electromechanical relays in the Robinson proved to be too slow, and a new machine called the Colossus was built from vacuum tubes. It was completed in 1943, and by the end of the war, ten Colossus machines were in everyday use.

The first operational *programmable* computer was the Z-3, the invention of Konrad Zuse in Germany in 1941. Zuse invented floating-point numbers for the Z-3, and went on in 1945 to develop Plankalkul, the first high-level programming language. Although Zuse received some support from the Third Reich to apply his machine to aircraft design, the military hierarchy did not attach as much importance to computing as did its counterpart in Britain.

In the United States, the first *electronic* computer, the ABC, was assembled by John Atanasoff and his graduate student Clifford Berry between 1940 and 1942 at Iowa State University. The project received little support and was abandoned after Atanasoff became involved in military research in Washington. Two other computer projects were started as secret military research: the Mark I, II, and III computers were developed at Harvard by a team under Howard Aiken; and the ENIAC was developed at the University of Pennsylvania by a team including John Mauchly and John Eckert. ENIAC was the first general-purpose, electronic, digital computer. One of its first applications was computing artillery firing tables. A successor, the EDVAC, followed John Von Neumann's suggestion to use a stored program, so that technicians would not have to scurry about changing patch cords to run a new program.

But perhaps the most critical breakthrough was the IBM 701, built in 1952 by Nathaniel Rochester and his group. This was the first computer to yield a profit for its manufacturer. IBM went on to become one of the world's largest corporations, and sales of computers have grown to $150 billion/year. In the United States, the computer industry (including software and services) now accounts for about 10% of the gross national product.

Each generation of computer hardware has brought an increase in speed and capacity, and a decrease in price. Computer engineering has been remarkably successful, regularly doubling performance every two years, with no immediate end in sight for this rate of increase. Massively parallel machines promise to add several more zeros to the overall throughput achievable.

Of course, there were calculating devices before the electronic computer. The abacus is roughly 7000 years old. In the mid-17th century, Blaise Pascal built a mechanical adding and subtracting machine called the Pascaline. Leibniz improved on this in 1694. building a mechanical device that multiplied by doing repeated addition. Progress stalled for over a century until Charles Babbage (1792–1871) dreamed that logarithm tables could be computed by machine. He designed a machine for this task, but never completed the project. Instead, he turned to the design of the Analytical Engine, for which Babbage invented the ideas of addressable memory, stored programs, and conditional jumps. Although the idea of programmable machines was not new—in 1805. Joseph Marie Jacquard invented a loom that could be programmed using punched cards—Babbage's machine was the first artifact possessing the characteristics necessary for universal computation. Babbage's colleague Ada Lovelace, daughter of the poet Lord Byron, wrote programs for the Analytical Engine and even speculated that the machine could play chess or compose music. Lovelace was the world's first programmer, and the first of many to endure massive cost overruns and to have an ambitious project ultimately abandoned.[11] Babbage's basic design was proven viable by Doron Swade and his colleagues, who built a working model using only the mechanical techniques available at Babbage's time (Swade. 1993). Babbage had the right idea, but lacked the organizational skills to get his machine built.

---

[10] Heath Robinson was a cartoonist famous for his depictions of whimsical and absurdly complicated contraptions for everyday tasks such as buttering toast.

AI also owes a debt to the software side of computer science, which has supplied the operating systems, programming languages, and tools needed to write modern programs (and papers about them). But this is one area where the debt has been repaid: work in AI has pioneered many ideas that have made their way back to "mainstream" computer science, including time sharing, interactive interpreters, the linked list data type, automatic storage management, and some of the key concepts of object-oriented programming and integrated program development environments with graphical user interfaces.

Linguistics (1957–present)

In 1957. B. F. Skinner published *Verbal Behavior.* This was a comprehensive, detailed account of the behaviorist approach to language learning, written by the foremost expert in the field. But curiously, a review of the book became as well-known as the book itself, and served to almost kill off interest in behaviorism. The author of the review was Noam Chomsky, who had just published a book on his own theory. *Syntactic Structures.* Chomsky showed how the behaviorist theory did not address the notion of creativity in language—it did not explain how a child could understand and make up sentences that he or she had never heard before. Chomsky's theory—based on syntactic models going back to the Indian linguist Panini (c. 350 B.C.)—could explain this, and unlike previous theories, it was formal enough that it could in principle be programmed.

Later developments in linguistics showed the problem to be considerably more complex than it seemed in 1957. Language is ambiguous and leaves much unsaid. This means that understanding language requires an understanding of the subject matter and context, not just an understanding of the structure of sentences. This may seem obvious, but it was not appreciated until the early 1960s. Much of the early work in **knowledge representation** (the study of how to put knowledge into a form that a computer can reason with) was tied to language and informed by research in linguistics, which was connected in turn to decades of work on the philosophical analysis of language.

---

[11] She also gave her name to Ada. the U.S. Department of Defense's all-purpose programming language.

Modern linguistics and AI were "born" at about the same time, so linguistics does not play a large foundational role in the growth of AI. Instead, the two grew up together, intersecting in a hybrid field called **computational linguistics or natural language processing,** which concentrates on the problem of language use.

1.3 THE HISTORY OF ARTIFICIAL INTELLIGENCE

With the background material behind us, we are now ready to outline the development of AI proper. We could do this by identifying loosely defined and overlapping phases in its development, or by chronicling the various different and intertwined conceptual threads that make up the field. In this section, we will take the former approach, at the risk of doing some degree of violence to the real relationships among subfields. The history of each subfield is covered in individual chapters later in the book.

The gestation of artificial intelligence (1943-1956)

The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943). They drew on three sources: knowledge of the basic physiology and function of neurons in the brain; the formal analysis of propositional logic due to Russell and Whitehead; and Turing's theory of computation. They proposed a model of artificial neurons in which each neuron is characterized as being "on" or "off," with a switch to "on" occurring in response to stimulation by a sufficient number of neighboring neurons. The state of a neuron was conceived of as "factually equivalent to a proposition which proposed its adequate stimulus." They showed, for example, that any computable function could be computed by some network of connected neurons, and that all the logical connectives could be implemented by simple net structures. McCulloch and Pitts also suggested that suitably defined networks could learn. Donald Hebb (1949) demonstrated a simple updating rule for modifying the connection strengths between neurons, such that learning could take place.

The work of McCulloch and Pitts was arguably the forerunner of both the logicist tradition in AI and the connectionist tradition. In the early 1950s, Claude Shannon (1950) and Alan Turing (1953) were writing chess programs for von Neumann-style conventional computers.[12] At the same time, two graduate students in the Princeton mathematics department, Marvin Minsky and Dean Edmonds, built the first neural network computer in 1951. The SNARC, as it was called, used 3000 vacuum tubes and a surplus automatic pilot mechanism from a B-24 bomber to simulate a network of 40 neurons. Minsky's Ph.D. committee was skeptical whether this kind of work should be considered mathematics, but von Neumann was on the committee and reportedly said, "If it isn't now it will be someday." Ironically, Minsky was later to prove theorems that contributed to the demise of much of neural network research during the 1970s.

---

[12] Shannon actually had no real computer to work with, and Turing was eventually denied access to his own team's computers by the British government, on the grounds that research into artificial intelligence was surely frivolous.

Princeton was home to another influential figure in AI, John McCarthy. After graduation, McCarthy moved to Dartmouth College, which was to become the official birthplace of the field. McCarthy convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence. They organized a two-month workshop at Dartmouth in the summer of 1956. All together there were ten attendees, including Trenchard More from Princeton, Arthur Samuel from IBM, and Ray Solomonoff and Oliver Selfridge from MIT.

Two researchers from Carnegie Tech,[13] Alien Newell and Herbert Simon, rather stole the show. Although the others had ideas and in some cases programs for particular applications such as checkers, Newell and Simon already had a reasoning program, the Logic Theorist (LT), about which Simon claimed, "We have invented a computer program capable of thinking non-numerically, and thereby solved the venerable mind-body problem."[14] Soon after the workshop, the program was able to prove most of the theorems in Chapter 2 of Russell and Whitehead's *Principia Mathematica.* Russell was reportedly delighted when Simon showed him that the program had come up with a proof for one theorem that was shorter than the one in *Principia.* The editors of the *Journal of Symbolic Logic* were less impressed; they rejected a paper coauthored by Newell, Simon, and Logic Theorist.

The Dartmouth workshop did not lead to any new breakthroughs, but it did introduce all the major figures to each other. For the next 20 years, the field would be dominated by these people and their students and colleagues at MIT, CMU, Stanford, and IBM. Perhaps the most lasting thing to come out of the workshop was an agreement to adopt McCarthy's new name for the field: **artificial intelligence.**

Early enthusiasm, great expectations (1952-1969)

The early years of AI were full of successes—in a limited way. Given the primitive computers and programming tools of the time, and the fact that only a few years earlier computers were seen as things that could do arithmetic and no more, it was astonishing whenever a computer did anything remotely clever. The intellectual establishment, by and large, preferred to believe that "a machine can never do *X*" (see Chapter 26 for a long list of *X's* gathered by Turing). AI researchers naturally responded by demonstrating one *X* after another. Some modern AI researchers refer to this period as the "Look, Ma, no hands!" era.

Newell and Simon's early success was followed up with the General Problem Solver, or GPS. Unlike Logic Theorist, this program was designed from the start to imitate human problem-solving protocols. Within the limited class of puzzles it could handle, it turned out that the order in which the program considered subgoals and possible actions was similar to the way humans approached the same problems. Thus, GPS was probably the first program to embody the "thinking humanly" approach. The combination of AI and cognitive science has continued at CMU up to the present day.

---

[13] Now Carnegie Mellon University (CMU).

[14] Newell and Simon also invented a list-processing language, IPL, to write LT. They had no compiler, and translated it into machine code by hand. To avoid errors, they worked in parallel, calling out binary numbers to each other as they wrote each instruction to make sure they agreed.

At IBM, Nathaniel Rochester and his colleagues produced some of the first AI programs. Herbert Gelernter (1959) constructed the Geometry Theorem Prover. Like the Logic Theorist, it proved theorems using explicitly represented axioms. Gelernter soon found that there were too many possible reasoning paths to follow, most of which turned out to be dead ends. To help focus the search, he added the capability to create a numerical representation of a diagram—a particular case of the general theorem to be proved. Before the program tried to prove something, it could first check the diagram to see if it was true in the particular case.

Starting in 1952, Arthur Samuel wrote a series of programs for checkers (draughts) that eventually learned to play tournament-level checkers. Along the way, he disproved the idea that computers can only do what they are told to, as his program quickly learned to play a better game than its creator. The program was demonstrated on television in February 1956, creating a very strong impression. Like Turing, Samuel had trouble finding computer time. Working at night, he used machines that were still on the testing floor at IBM's manufacturing plant. Chapter 5 covers game playing, and Chapter 20 describes and expands on the learning techniques used by Samuel.

LISP

John McCarthy moved from Dartmouth to MIT and there made three crucial contributions in one historic year: 1958. In MIT AI Lab Memo No. 1, McCarthy defined the high-level language **Lisp,** which was to become the dominant AI programming language. Lisp is the second-oldest language in current use.[15] With Lisp, McCarthy had the tool he needed, but access to scarce and expensive computing resources was also a serious problem. Thus, he and others at MIT invented time sharing. After getting an experimental time-sharing system up at MIT, McCarthy eventually attracted the interest of a group of MIT grads who formed Digital Equipment Corporation, which was to become the world's second largest computer manufacturer, thanks to their time-sharing minicomputers. Also in 1958, McCarthy published a paper entitled *Programs with Common Sense*, in which he described the Advice Taker, a hypothetical program that can be seen as the first complete AI system. Like the Logic Theorist and Geometry Theorem Prover, McCarthy's program was designed to use knowledge to search for solutions to problems. But unlike the others, it was to embody general knowledge of the world. For example, he showed how some simple axioms would enable the program to generate a plan to drive to the airport to catch a plane. The program was also designed so that it could accept new axioms in the normal course of operation, thereby allowing it to achieve competence in new areas *without being reprogrammed.* The Advice Taker thus embodied the central principles of knowledge representation and reasoning: that it is useful to have a formal, explicit representation of the world and the way an agent's actions affect the world, and to be able to manipulate these representations with deductive processes. It is remarkable how much of the 1958 paper remains relevant after more than 35 years.

1958 also marked the year that Marvin Minsky moved to MIT. For years he and McCarthy were inseparable as they defined the field together. But they grew apart as McCarthy stressed representation and reasoning in formal logic, whereas Minsky was more interested in getting programs to work, and eventually developed an anti-logical outlook. In 1963, McCarthy took the opportunity to go to Stanford and start the AI lab there. His research agenda of using logic to build the ultimate Advice Taker was advanced by J. A. Robinson's discovery of the resolution method (a complete theorem-proving algorithm for first-order logic; see Section 9.6). Work at Stanford emphasized general-purpose methods for logical reasoning. Applications of logic included Cordell Green's question answering and planning systems (Green, 1969b), and the Shakey robotics project at the new Stanford Research Institute (SRI). The latter project, discussed further in Chapter 25, was the first to demonstrate the complete integration of logical reasoning and physical activity.

---

[15] FORTRAN is one year older than Lisp.

MICROWORLDS

Minsky supervised a series of students who chose limited problems that appeared to require intelligence to solve. These limited domains became known as **microworlds**. James Slagle's SAINT program (1963a) was able to solve closed-form integration problems typical of first-year college calculus courses. Tom Evans's ANALOGY program (1968) solved geometric analogy problems that appear in IQ tests, such as the one in Figure 1.2. Bertram Raphael's (1968) SIR (Semantic Information Retrieval) was able to accept input statements in a very restricted subset of English and answer questions thereon. Daniel Bobrow's STUDENT program (1967) solved algebra story problems such as

> If the number of customers Tom gets is twice the square of 20 percent of the number of advertisements he runs, and the number of advertisements he runs is 45, what is the number of customers Tom gets?

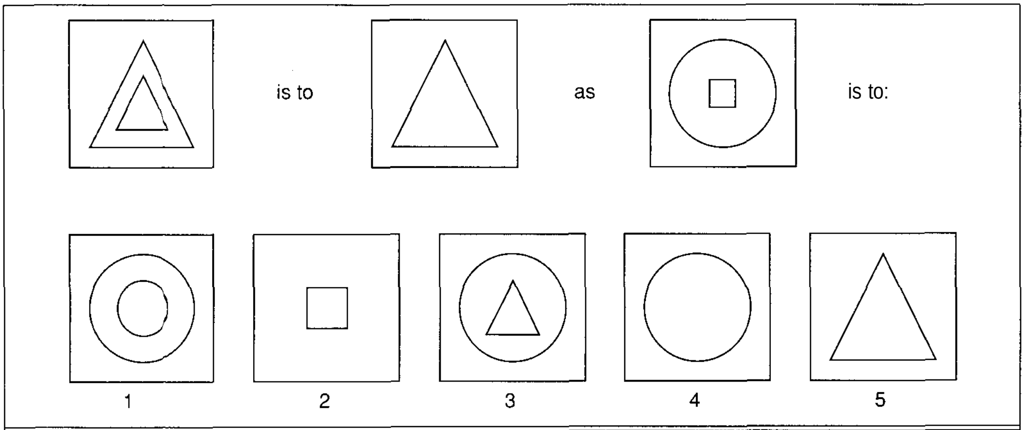

Figure 1.2 An example problem solved by Evans's ANALOGY program.

The most famous microworld was the blocks world, which consists of a set of solid blocks placed on a tabletop (or more often, a simulation of a tabletop), as shown in Figure 1.3. A task in this world is to rearrange the blocks in a certain way, using a robot hand that can pick up one block at a time. The blocks world was home to the vision project of David Huffman (1971), the vision and constraint-propagation work of David Waltz (1975), the learning theory of Patrick Winston (1970), the natural language understanding program of Terry Winograd (1972), and the planner of Scott Fahlman (1974).

Early work building on the neural networks of McCulloch and Pitts also flourished. The work of Winograd and Cowan (1963) showed how a large number of elements could collectively represent an individual concept, with a corresponding increase in robustness and parallelism. Hebb's learning methods were enhanced by Bernie Widrow (Widrow and Hoff, 1960; Widrow, 1962), who called his networks **adalines,** and by Frank Rosenblatt (1962) with his **perceptrons**.

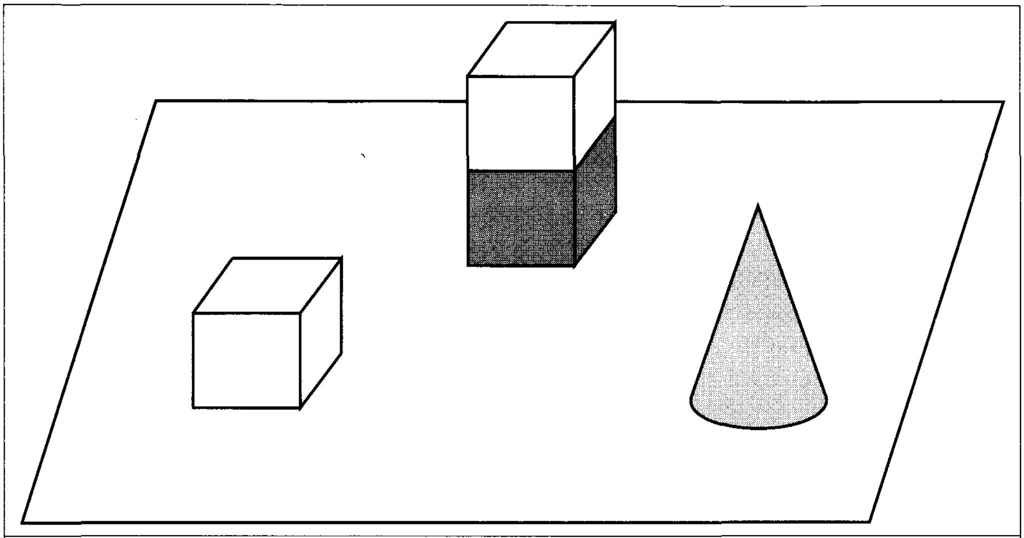

Figure 1.3 A scene from the blocks world. A task for the robot might be "Pick up a big red block," expressed either in natural language or in a formal notation.

Rosenblatt proved the famous **perceptron convergence theorem,** showing that his learning algorithm could adjust the connection strengths of a perceptron to match any input data, provided such a match existed. These topics are covered in Section 19.3.

A dose of reality (1966-1974)

From the beginning, AI researchers were not shy in making predictions of their coming successes. The following statement by Herbert Simon in 1957 is often quoted:

> It is not my aim to surprise or shock you—but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until—in a visible future—the range of problems they can handle will be coextensive with the range to which human mind has been applied.

Although one might argue that terms such as "visible future" can be interpreted in various ways, some of Simon's predictions were more concrete. In 1958, he predicted that within 10 years a computer would be chess champion, and an important new mathematical theorem would be proved by machine. Claims such as these turned out to be wildly optimistic. The barrier that faced almost all AI research projects was that methods that sufficed for demonstrations on one or two simple examples turned out to fail miserably when tried out on wider selections of problems and on more difficult problems.

The first kind of difficulty arose because early programs often contained little or no knowledge of their subject matter, and succeeded by means of simple syntactic manipulations. Weizenbaum's ELIZA program (1965), which could apparently engage in serious conversation on any topic, actually just borrowed and manipulated the sentences typed into it by a human. A typical story occurred in early machine translation efforts, which were generously funded by the National Research Council in an attempt to speed up the translation of Russian scientific papers in the wake of the Sputnik launch in 1957. It was thought initially that simple syntactic transformations based on the grammars of Russian and English, and word replacement using an electronic dictionary, would suffice to preserve the exact meanings of sentences. In fact, translation requires general knowledge of the subject matter in order to resolve ambiguity and establish the content of the sentence. The famous retranslation of "the spirit is willing but the flesh is weak" as "the vodka is good but the meat is rotten" illustrates the difficulties encountered. In 1966, a report by an advisory committee found that "there has been no machine translation of general scientific text, and none is in immediate prospect." All U.S. government funding for academic translation projects was cancelled.

The second kind of difficulty was the intractability of many of the problems that AI was attempting to solve. Most of the early AI programs worked by representing the basic facts about a problem and trying out a series of steps to solve it, combining different combinations of steps until the right one was found. The early programs were feasible only because microworlds contained very few objects. Before the theory of NP-completeness was developed, it was widely thought that "scaling up" to larger problems was simply a matter of faster hardware and larger memories. The optimism that accompanied the development of resolution theorem proving, for example, was soon dampened when researchers failed to prove theorems involving more than a few dozen facts. *The fact that a program can find a solution in principle does not mean that the program contains any of the mechanisms needed to find it in practice.*

MACHINE EVOLUTION

The illusion of unlimited computational power was not confined to problem-solving programs. Early experiments in **machine evolution** (now called **genetic algorithms**) (Friedberg, 1958; Friedberg *et al.*, 1959) were based on the undoubtedly correct belief that by making an appropriate series of small mutations to a machine code program, one can generate a program with good performance for any particular simple task. The idea, then, was to try random mutations and then apply a selection process to preserve mutations that seemed to improve behavior. Despite thousands of hours of CPU time, almost no progress was demonstrated.

Failure to come to grips with the "combinatorial explosion" was one of the main criticisms of AI contained in the Lighthill report (Lighthill, 1973), which formed the basis for the decision by the British government to end support for AI research in all but two universities. (Oral tradition paints a somewhat different and more colorful picture, with political ambitions and personal animosities that cannot be put in print.)

A third difficulty arose because of some fundamental limitations on the basic structures being used to generate intelligent behavior. For example, in 1969, Minsky and Papert's book *Perceptrons* (1969) proved that although perceptrons could be shown to learn anything they were capable of representing, they could represent very little. In particular, a two-input perceptron could not be trained to recognize when its two inputs were different. Although their results did not apply to more complex, multilayer networks, research funding for neural net research soon dwindled to almost nothing. Ironically, the new back-propagation learning algorithms for multilayer networks that were to cause an enormous resurgence in neural net research in the late 1980s were actually discovered first in 1969 (Bryson and Ho, 1969).

Knowledge-based systems: The key to power? (1969-1979)

WEAK METHODS

The picture of problem solving that had arisen during the first decade of AI research was of a general-purpose search mechanism trying to string together elementary reasoning steps to find complete solutions. Such approaches have been called **weak methods,** because they use weak information about the domain. For many complex domains, it turns out that their performance is also weak. The only way around this is to use knowledge more suited to making larger reasoning steps and to solving typically occurring cases in narrow areas of expertise. One might say that to solve a hard problem, you almost have to know the answer already.

The DENDRAL program (Buchanan *et al.*, 1969) was an early example of this approach. It was developed at Stanford, where Ed Feigenbaum (a former student of Herbert Simon), Bruce Buchanan (a philosopher turned computer scientist), and Joshua Lederberg (a Nobel laureate geneticist) teamed up to solve the problem of inferring molecular structure from the information provided by a mass spectrometer. The input to the program consists of the elementary formula of the molecule (e.g., $C_6H_{13}NO_2$), and the mass spectrum giving the masses of the various fragments of the molecule generated when it is bombarded by an electron beam. For example, the mass spectrum might contain a peak at $m = 15$ corresponding to the mass of a methyl (CH_3) fragment.

The naive version of the program generated all possible structures consistent with the formula, and then predicted what mass spectrum would be observed for each, comparing this with the actual spectrum. As one might expect, this rapidly became intractable for decent-sized molecules. The DENDRAL researchers consulted analytical chemists and found that they worked by looking for well-known patterns of peaks in the spectrum that suggested common substructures in the molecule. For example, the following rule is used to recognize a ketone (C=O) subgroup:

```
if there are two peaks at $x_1$ and $x_2$ such that
(a) $x_1 + x_2 = M + 28$ ($M$ is the mass of the whole molecule);
(b) $x_1 - 28$ is a high peak;
(c) $x_2 - 28$ is a high peak;
(d) At least one of $x_1$ and $x_2$ is high.
then there is a ketone subgroup
```

Having recognized that the molecule contains a particular substructure, the number of possible candidates is enormously reduced. The DENDRAL team concluded that the new system was powerful because

> All the relevant theoretical knowledge to solve these problems has been mapped over from its general form in the [spectrum prediction component] ("first principles") to efficient special forms ("cookbook recipes"). (Feigenbaum *et al.*, 1971)

The significance of DENDRAL was that it was arguably the first successful *knowledge-intensive* system: its expertise derived from large numbers of special-purpose rules. Later systems also incorporated the main theme of McCarthy's Advice Taker approach—the clean separation of the knowledge (in the form of rules) and the reasoning component.

EXPERT SYSTEMS

With this lesson in mind, Feigenbaum and others at Stanford began the Heuristic Programming Project (HPP), to investigate the extent to which the new methodology of **expert systems** could be applied to other areas of human expertise. The next major effort was in the area of medical diagnosis. Feigenbaum, Buchanan, and Dr. Edward Shortliffe developed MYCIN to diagnose blood infections. With about 450 rules, MYCIN was able to perform as well as some experts, and considerably better than junior doctors. It also contained two major differences from DENDRAL. First, unlike the DENDRAL rules, no general theoretical model existed from which the MYCIN rules could be deduced. They had to be acquired from extensive interviewing of experts, who in turn acquired them from direct experience of cases. Second, the rules had to reflect the uncertainty associated with medical knowledge. MYCIN incorporated a calculus of uncertainty called **certainty factors** (see Chapter 14), which seemed (at the time) to fit well with how doctors assessed the impact of evidence on the diagnosis.

Other approaches to medical diagnosis were also followed. At Rutgers University, Saul Amarel's *Computers in Biomedicine* project began an ambitious attempt to diagnose diseases based on explicit knowledge of the causal mechanisms of the disease process. Meanwhile, large groups at MIT and the New England Medical Center were pursuing an approach to diagnosis and treatment based on the theories of probability and utility. Their aim was to build systems that gave provably optimal medical recommendations. In medicine, the Stanford approach using rules provided by doctors proved more popular at first. But another probabilistic reasoning system, PROSPECTOR (Duda *et al.*, 1979), generated enormous publicity by recommending exploratory drilling at a geological site that proved to contain a large molybdenum deposit.

The importance of domain knowledge was also apparent in the area of understanding natural language. Although Winograd's SHRDLU system for understanding natural language had engendered a good deal of excitement, its dependence on syntactic analysis caused some of the same problems as occurred in the early machine translation work. It was able to overcome ambiguity and understand pronoun references, but this was mainly because it was designed specifically for one area—the blocks world. Several researchers, including Eugene Charniak, a fellow graduate student of Winograd's at MIT, suggested that robust language understanding would require general knowledge about the world and a general method for using that knowledge.

At Yale, the linguist-turned-AI-researcher Roger Schank emphasized this point by claiming, "There is no such thing as syntax," which upset a lot of linguists, but did serve to start a useful discussion. Schank and his students built a series of programs (Schank and Abelson, 1977; Schank and Riesbeck, 1981; Dyer, 1983) that all had the task of understanding natural language. The emphasis, however, was less on language *per se* and more on the problems of representing and reasoning with the knowledge required for language understanding. The problems included representing stereotypical situations (Cullingford, 1981), describing human memory organization (Rieger, 1976; Kolodner, 1983), and understanding plans and goals (Wilensky, 1983). William Woods (1973) built the LUNAR system, which allowed geologists to ask questions in English about the rock samples brought back by the Apollo moon mission. LUNAR was the first natural language program that was used by people other than the system's author to get real work done. Since then, many natural language programs have been used as interfaces to databases.

The widespread growth of applications to real-world problems caused a concomitant increase in the demands for workable knowledge representation schemes. A large number of different representation languages were developed. Some were based on logic—for example, the Prolog language became popular in Europe, and the PLANNER family in the United States. Others, following Minsky's idea of **frames** (1975), adopted a rather more structured approach, collecting together facts about particular object and event types, and arranging the types into a large taxonomic hierarchy analogous to a biological taxonomy.

FRAMES

AI becomes an industry (1980-1988)

The first successful commercial expert system, R1, began operation at Digital Equipment Corporation (McDermott, 1982). The program helped configure orders for new computer systems, and by 1986, it was saving the company an estimated $40 million a year. By 1988, DEC's AI group had 40 deployed expert systems, with more on the way. Du Pont had 100 in use and 500 in development, saving an estimated $10 million a year. Nearly every major U.S. corporation had its own AI group and was either using or investigating expert system technology.

In 1981, the Japanese announced the "Fifth Generation" project, a 10-year plan to build intelligent computers running Prolog in much the same way that ordinary computers run machine code. The idea was that with the ability to make millions of inferences per second, computers would be able to take advantage of vast stores of rules. The project proposed to achieve full-scale natural language understanding, among other ambitious goals.

The Fifth Generation project fueled interest in AI, and by taking advantage of fears of Japanese domination, researchers and corporations were able to generate support for a similar investment in the United States. The Microelectronics and Computer Technology Corporation (MCC) was formed as a research consortium to counter the Japanese project. In Britain, the Alvey report reinstated the funding that was cut by the Lighthill report.[16] In both cases, AI was part of a broad effort, including chip design and human-interface research.

The booming AI industry also included companies such as Carnegie Group, Inference, Intellicorp, and Teknowledge that offered the software tools to build expert systems, and hardware companies such as Lisp Machines Inc., Texas Instruments, Symbolics, and Xerox that were building workstations optimized for the development of Lisp programs. Over a hundred companies built industrial robotic vision systems. Overall, the industry went from a few million in sales in 1980 to $2 billion in 1988.

The return of neural networks (1986–present)

Although computer science had neglected the field of neural networks after Minsky and Papert's *Perceptrons* book, work had continued in other fields, particularly physics. Large collections of simple neurons could be understood in much the same way as large collections of atoms in solids. Physicists such as Hopfield (1982) used techniques from statistical mechanics to analyze the storage and optimization properties of networks, leading to significant cross-fertilization of ideas. Psychologists including David Rumelhart and Geoff Hinton continued the study of neural net models of memory. As we discuss in Chapter 19, the real impetus came in the mid-1980s when at least four different groups reinvented the back-propagation learning algorithm first found in 1969 by Bryson and Ho. The algorithm was applied to many learning problems in computer science and psychology, and the widespread dissemination of the results in the collection *Parallel Distributed Processing* (Rumelhart and McClelland, 1986) caused great excitement.

At about the same time, some disillusionment was occurring concerning the applicability of the expert system technology derived from MYCIN-type systems. Many corporations and research groups found that building a successful expert system involved much more than simply buying a reasoning system and filling it with rules. Some predicted an "AI Winter" in which AI funding would be squeezed severely. It was perhaps this fear, and the historical factors on the neural network side, that led to a period in which neural networks and traditional AI were seen as rival fields, rather than as mutually supporting approaches to the same problem.

---

[16] To save embarrassment, a new field called IKBS (Intelligent Knowledge-Based Systems) was defined because Artificial Intelligence had been officially cancelled.

Recent events (1987–present)

Recent years have seen a sea change in both the content and the methodology of research in artificial intelligence.[17] It is now more common to build on existing theories than to propose brand new ones, to base claims on rigorous theorems or hard experimental evidence rather than on intuition, and to show relevance to real-world applications rather than toy examples.

The field of speech recognition illustrates the pattern. In the 1970s, a wide variety of different architectures and approaches were tried. Many of these were rather *ad hoc* and fragile, and were demonstrated on a few specially selected examples. In recent years, approaches based on **hidden Markov models** (HMMs) have come to dominate the area. Two aspects of HMMs are relevant to the present discussion. First, they are based on a rigorous mathematical theory. This has allowed speech researchers to build on several decades of mathematical results developed in other fields. Second, they are generated by a process of training on a large corpus of real speech data. This ensures that the performance is robust, and in rigorous blind tests the HMMs have been steadily improving their scores. Speech technology and the related field of handwritten character recognition are already making the transition to widespread industrial and consumer applications.

Another area that seems to have benefitted from formalization is planning. Early work by Austin Tate (1977), followed up by David Chapman (1987), has resulted in an elegant synthesis of existing planning programs into a simple framework. There have been a number of advances that built upon each other rather than starting from scratch each time. The result is that planning systems that were only good for microworlds in the 1970s are now used for scheduling of factory work and space missions, among other things. See Chapters 11 and 12 for more details.

Judea Pearl's (1988) *Probabilistic Reasoning in Intelligent Systems* marked a new acceptance of probability and decision theory in AI, following a resurgence of interest epitomized by Peter Cheeseman's (1985) article "In Defense of Probability." The **belief network** formalism was invented to allow efficient reasoning about the combination of uncertain evidence. This approach largely overcomes the problems with probabilistic reasoning systems of the 1960s and 1970s, and has come to dominate AI research on uncertain reasoning and expert systems. Work by Judea Pearl (1982a) and by Eric Horvitz and David Heckerman (Horvitz and Heckerman, 1986; Horvitz *et al.*, 1986) promoted the idea of *normative* expert systems: ones that act rationally according to the laws of decision theory and do not try to imitate human experts. Chapters 14 to 16 cover this area.

---

[17] Some have characterized this change as a victory of the **neats**—those who think that AI theories should be grounded in mathematical rigor—over the **scruffies**—those who would rather try out lots of ideas, write some programs, and then assess what seems to be working. Both approaches are important. A shift toward increased neatness implies that the field has reached a level of stability and maturity. (Whether that stability will be disrupted by a new scruffy idea is another question.)

Similar gentle revolutions have occurred in robotics, computer vision, machine learning (including neural networks), and knowledge representation. A better understanding of the problems and their complexity properties, combined with increased mathematical sophistication, has led to workable research agendas and robust methods. Perhaps encouraged by the progress in solving the subproblems of AI, researchers have also started to look at the "whole agent" problem again. The work of Alien Newell, John Laird, and Paul Rosenbloom on SOAR (Newell, 1990; Laird *et al.*, 1987) is the best-known example of a complete agent architecture in AI. The so-called "situated" movement aims to understand the workings of agents embedded in real environments with continuous sensory inputs. Many interesting results are coming out of such work, including the realization that the previously isolated subfields of AI may need to be reorganized somewhat when their results are to be tied together into a single agent design.

1.4 THE STATE OF THE ART

International grandmaster Arnold Denker studies the pieces on the board in front of him. He realizes there is no hope; he must resign the game. His opponent, HITECH, becomes the first computer program to defeat a grandmaster in a game of chess (Berliner, 1989).

"I want to go from Boston to San Francisco," the traveller says into the microphone. "What date will you be travelling on?" is the reply. The traveller explains she wants to go October 20th, nonstop, on the cheapest available fare, returning on Sunday. A speech understanding program named PEGASUS handles the whole transaction, which results in a confirmed reservation that saves the traveller $894 over the regular coach fare. Even though the speech recognizer gets one out of ten words wrong,[18] it is able to recover from these errors because of its understanding of how dialogs are put together (Zue *et al.*, 1994).

An analyst in the Mission Operations room of the Jet Propulsion Laboratory suddenly starts paying attention. A red message has flashed onto the screen indicating an "anomaly" with the Voyager spacecraft, which is somewhere in the vicinity of Neptune. Fortunately, the analyst is able to correct the problem from the ground. Operations personnel believe the problem might have been overlooked had it not been for MARVEL, a real-time expert system that monitors the massive stream of data transmitted by the spacecraft, handling routine tasks and alerting the analysts to more serious problems (Schwuttke, 1992).

Cruising the highway outside of Pittsburgh at a comfortable 55 mph, the man in the driver's seat seems relaxed. He should be—for the past 90 miles, he has not had to touch the steering wheel, brake, or accelerator. The real driver is a robotic system that gathers input from video cameras, sonar, and laser range finders attached to the van. It combines these inputs with experience learned from training runs and succesfully computes how to steer the vehicle (Pomerleau, 1993).

A leading expert on lymph-node pathology describes a fiendishly difficult case to the expert system, and examines the system's diagnosis. He scoffs at the system's response. Only slightly worried, the creators of the system suggest he ask the computer for an explanation of the diagnosis. The machine points out the major factors influencing its decision, and explains the subtle interaction of several of the symptoms in this case. The expert admits his error, eventually (Heckerman, 1991).

---

[18] Some other existing systems err only half as often on this task.

From a camera perched on a street light above the crossroads, the traffic monitor watches the scene. If any humans were awake to read the main screen, they would see "Citroen 2CV turning from Place de la Concorde into Champs Elysees," "Large truck of unknown make stopped on Place de la Concorde," and so on into the night. And occasionally, "Major incident on Place de la Concorde, speeding van collided with motorcyclist," and an automatic call to the emergency services (King *et al.*, 1993; Koller *et al.*, 1994).

These are just a few examples of artificial intelligence systems that exist today. Not magic or science fiction—but rather science, engineering, and mathematics, to which this book provides an introduction.

1.5 SUMMARY

This chapter defines AI and establishes the cultural background against which it has developed. Some of the important points are as follows:

- Different people think of AI differently. Two important questions to ask are: Are you concerned with thinking or behavior? Do you want to model humans, or work from an ideal standard?
- In this book, we adopt the view that intelligence is concerned mainly with **rational action.** Ideally, an **intelligent agent** takes the best possible action in a situation. We will study the problem of building agents that are intelligent in this sense.
- Philosophers (going back to 400 B.C.) made AI conceivable by considering the ideas that the mind is in some ways like a machine, that it operates on knowledge encoded in some internal language, and that thought can be used to help arrive at the right actions to take.
- Mathematicians provided the tools to manipulate statements of logical certainty as well as uncertain, probabilistic statements. They also set the groundwork for reasoning about algorithms.
- Psychologists strengthened the idea that humans and other animals can be considered information processing machines. Linguists showed that language use fits into this model.
- Computer engineering provided the artifact that makes AI applications possible. AI programs tend to be large, and they could not work without the great advances in speed and memory that the computer industry has provided.
- The history of AI has had cycles of success, misplaced optimism, and resulting cutbacks in enthusiasm and funding. There have also been cycles of introducing new creative approaches and systematically refining the best ones.
- Recent progress in understanding the theoretical basis for intelligence has gone hand in hand with improvements in the capabilities of real systems.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Daniel Crevier's (1993) *Artificial Intelligence* gives a complete history of the field, and Raymond Kurzweil's (1990) *Age of Intelligent Machines* situates AI in the broader context of computer science and intellectual history in general. Dianne Martin (1993) documents the degree to which early computers were endowed by the media with mythical powers of intelligence.

The methodological status of artificial intelligence is discussed in *The Sciences of the Artificial,* by Herb Simon (1981), which discusses research areas concerned with complex artifacts. It explains how AI can be viewed as both science and mathematics.

Artificial Intelligence: The Very Idea, by John Haugeland (1985) gives a readable account of the philosophical and practical problems of AI. Cognitive science is well-described by Johnson-Laird's *The Computer and the Mind: An Introduction to Cognitive Science.* Baker (1989) covers the syntactic part of modern linguistics, and Chierchia and McConnell-Ginet (1990) cover semantics. Alien (1995) covers linguistics from the AI point of view.

Early AI work is covered in Feigenbaum and Feldman's *Computers and Thought,* Minsky's *Semantic Information Processing,* and the *Machine Intelligence* series edited by Donald Michie. A large number of influential papers are collected in *Readings in Artificial Intelligence* (Webber and Nilsson, 1981). Early papers on neural networks are collected in *Neurocomputing* (Anderson and Rosenfeld, 1988). The *Encyclopedia of AI* (Shapiro, 1992) contains survey articles on almost every topic in AI. These articles usually provide a good entry point into the research literature on each topic. The four-volume *Handbook of Artificial Intelligence* (Barr and Feigenbaum, 1981) contains descriptions of almost every major AI system published before 1981.

The most recent work appears in the proceedings of the major AI conferences: the biennial International Joint Conference on AI (IJCAI), and the annual National Conference on AI, more often known as AAAI, after its sponsoring organization. The major journals for general AI are *Artificial Intelligence, Computational Intelligence,* the IEEE *Transactions on Pattern Analysis and Machine Intelligence,* and the electronic *Journal of Artificial Intelligence Research.* There are also many journals devoted to specific areas, which we cover in the appropriate chapters. Commercial products are covered in the magazines *AI Expert* and *PC AI.* The main professional societies for AI are the American Association for Artificial Intelligence (AAAI), the ACM Special Interest Group in Artificial Intelligence (SIGART), and the Society for Artificial Intelligence and Simulation of Behaviour (AISB). AAAI's *AI Magazine* and the *SIGART Bulletin* contain many topical and tutorial articles as well as announcements of conferences and workshops.

EXERCISES

These exercises are intended to stimulate discussion, and some might be set as term projects. Alternatively, preliminary attempts can be made now, and these attempts can be reviewed after completing the book.

1.1 Read Turing's original paper on AI (Turing, 1950). In the paper, he discusses several potential objections to his proposed enterprise and his test for intelligence. Which objections still carry some weight? Are his refutations valid? Can you think of new objections arising from developments since he wrote the paper? In the paper, he predicts that by the year 2000, a computer will have a 30% chance of passing a five-minute Turing Test with an unskilled interrogator. Do you think this is reasonable?

1.2 We characterized the definitions of AI along two dimensions, human vs. ideal and thought vs. action. But there are other dimensions that are worth considering. One dimension is whether we are interested in theoretical results or in practical applications. Another is whether we intend our intelligent computers to be conscious or not. Philosophers have had a lot to say about this issue, and although most AI researchers are happy to leave the questions to the philosophers, STRONG AI there has been heated debate. The claim that machines can be conscious is called the strong AI WEAK AI claim; the **weak AI** position makes no such claim. Characterize the eight definitions on page 5 and the seven following definitions according to the four dimensions we have mentioned and whatever other ones you feel are helpful.

Artificial intelligence is ...

a. "a collection of algorithms that are computationally tractable, adequate approximations of intractably specified problems" (Partridge, 1991)

b. "the enterprise of constructing a physical symbol system that can reliably pass the Turing Test" (Ginsberg, 1993)

c. "the field of computer science that studies how machines can be made to act intelligently" (Jackson, 1986)

d. "a field of study that encompasses computational techniques for performing tasks that apparently require intelligence when performed by humans" (Tanimoto, 1990)

e. "a very general investigation of the nature of intelligence and the principles and mechanisms required for understanding or replicating it" (Sharpies *et al.*, 1989)

f. "the getting of computers to do things that seem to be intelligent" (Rowe, 1988).

1.3 There are well-known classes of problems that are intractably difficult for computers, and other classes that are provably undecidable by any computer. Does this mean that AI is impossible?

1.4 Suppose we extend Evans's ANALOGY program so that it can score 200 on a standard IQ test. Would we then have a program more intelligent than a human? Explain.

1.5 Examine the AI literature to discover whether or not the following tasks can currently be solved by computers:

a. Playing a decent game of table tennis (ping-pong).

b. Driving in the center of Cairo.

c. Playing a decent game of bridge at a competitive level.

d. Discovering and proving new mathematical theorems.

e. Writing an intentionally funny story.

f. Giving competent legal advice in a specialized area of law.

g. Translating spoken English into spoken Swedish in real time.

For the currently infeasible tasks. try to find out what the difficulties are and estimate when they will be overcome.

1.6 Find an article written by a lay person in a reputable newspaper or magazine claiming the achievement of some intelligent capacity by a machine, where the claim is either wildly exaggerated or false.

1.7 Fact, fiction, and forecast:

a. Find a claim in print by a reputable philosopher or scientist to the effect that a certain capacity will never be exhibited by computers, where that capacity has now been exhibited.
b. Find a claim by a reputable computer scientist to the effect that a certain capacity would be exhibited by a date that has since passed, without the appearance of that capacity.
c. Compare the accuracy of these predictions to predictions in other fields such as biomedicine, fusion power, nanotechnology, transportation, or home electronics.

1.8 Some authors have claimed that perception and motor skills are the most important part of intelligence, and that "higher-level" capacities are necessarily parasitic—simple add-ons to these underlying facilities. Certainly, most of evolution and a large part of the brain have been devoted to perception and motor skills, whereas AI has found tasks such as game playing and logical inference to be easier, in many ways, than perceiving and acting in the real world. Do you think that AI's traditional focus on higher-level cognitive abilities is misplaced?

1.9 "Surely computers cannot be intelligent—they can only do what their programmers tell them." Is the latter statement true, and does it imply the former?

1.10 "Surely animals cannot be intelligent—they can only do what their genes tell them." Is the latter statement true, and does it imply the former?

2 INTELLIGENT AGENTS

In which we discuss what an intelligent agent does, how it is related to its environment, how it is evaluated, and how we might go about building one.

2.1 INTRODUCTION

An agent is anything that can be viewed as **perceiving** its environment through **sensors and acting** upon that environment through **effectors.** A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors. A software agent has encoded bit strings as its percepts and actions. A generic agent is diagrammed in Figure 2.1.

Our aim in this book is to design agents that do a good job of acting on their environment. First, we will be a little more precise about what we mean by a good job. Then we will talk about different designs for successful agents—filling in the question mark in Figure 2.1. We discuss some of the general principles used in the design of agents throughout the book, chief among which is the principle that agents should *know* things. Finally, we show how to couple an agent to an environment and describe several kinds of environments.

2.2 How AGENTS SHOULD ACT

RATIONAL AGENT

A **rational agent** is one that does the right thing. Obviously, this is better than doing the wrong thing, but what does it mean? As a first approximation, we will say that the right action is the one that will cause the agent to be most successful. That leaves us with the problem of deciding *how* and *when* to evaluate the agent's success.

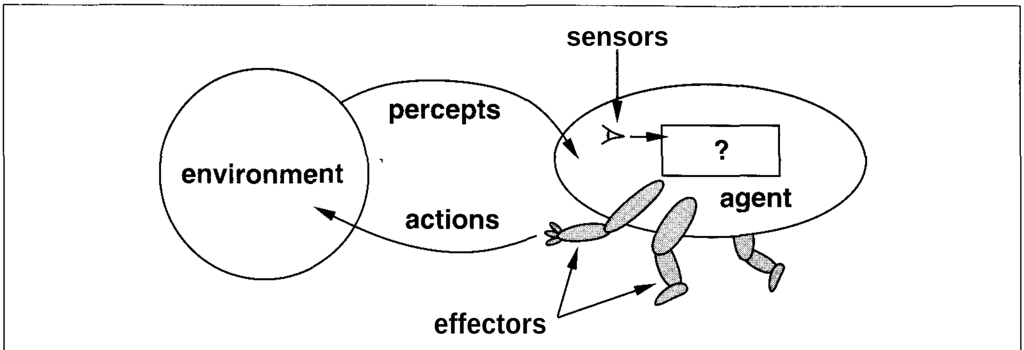

Figure 2.1 Agents interact with environments through sensors and effectors.

PERFORMANCE MEASURE

We use the term **performance measure** for the *how*—the criteria that determine how successful an agent is. Obviously, there is not one fixed measure suitable for all agents. We could ask the agent for a subjective opinion of how happy it is with its own performance, but some agents would be unable to answer, and others would delude themselves. (Human agents in particular are notorious for "sour grapes"—saying they did not really want something after they are unsuccessful at getting it.) Therefore, we will insist on an objective performance measure imposed by some authority. In other words, we as outside observers establish a standard of what it means to be successful in an environment and use it to measure the performance of agents.

As an example, consider the case of an agent that is supposed to vacuum a dirty floor. A plausible performance measure would be the amount of dirt cleaned up in a single eight-hour shift. A more sophisticated performance measure would factor in the amount of electricity consumed and the amount of noise generated as well. A third performance measure might give highest marks to an agent that not only cleans the floor quietly and efficiently, but also finds time to go windsurfing at the weekend.[1]

The *when* of evaluating performance is also important. If we measured how much dirt the agent had cleaned up in the first hour of the day, we would be rewarding those agents that start fast (even if they do little or no work later on), and punishing those that work consistently. Thus, we want to measure performance over the long run, be it an eight-hour shift or a lifetime.

OMNISCIENCE

We need to be careful to distinguish between rationality and **omniscience.** An omniscient agent knows the *actual* outcome of its actions, and can act accordingly; but omniscience is impossible in reality. Consider the following example: I am walking along the Champs Elysées one day and I see an old friend across the street. There is no traffic nearby and I'm not otherwise engaged, so, being rational, I start to cross the street. Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner,[2] and before I make it to the other side of the street I am flattened. Was I irrational to cross the street? It is unlikely that my obituary would read "Idiot attempts to cross street." Rather, this points out that rationality is concerned with *expected* success *given what has been perceived.* Crossing the street was rational because most of the time the crossing would be successful, and there was no way I could have foreseen the falling door. Note that another agent that was equipped with radar for detecting falling doors or a steel cage strong enough to repel them would be more successful, but it would not be any more rational.

---

[1] There is a danger here for those who establish performance measures: you often get what you ask for. That is. if you measure success by the amount of dirt cleaned up, then some clever agent is bound to bring in a load of dirt each morning, quickly clean it up, and get a good performance score. What you really want to measure is how clean the floor is, but determining that is more difficult than just weighing the dirt cleaned up.

[2] See N. Henderson, "New door latches urged for Boeing 747 jumbo jets." *Washington Post,* 8/24/89.

In other words, we cannot blame an agent for failing to take into account something it could not perceive, or for failing to take an action (such as repelling the cargo door) that it is incapable of taking. But relaxing the requirement of perfection is not just a question of being fair to agents. The point is that if we specify that an intelligent agent should always do what is *actually* the right thing, it will be impossible to design an agent to fulfill this specification—unless we improve the performance of crystal balls.

In summary, what is rational at any given time depends on four things:

PERCEPT SEQUENCE

- The performance measure that defines degree of success.
- Everything that the agent has perceived so far. We will call this complete perceptual history the **percept sequence.**
- What the agent knows about the environment.
- The actions that the agent can perform.

IDEAL RATIONAL AGENT

This leads to a definition of an **ideal rational agent:** *For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

We need to look carefully at this definition. At first glance, it might appear to allow an agent to indulge in some decidedly underintelligent activities. For example, if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed. The definition seems to say that it would be OK for it to cross the road. In fact, this interpretation is wrong on two counts. First, it would not be rational to cross the road: the risk of crossing without looking is too great. Second, an ideal rational agent would have chosen the "looking" action before stepping into the street, because looking helps maximize the expected performance. Doing actions *in order to obtain useful information* is an important part of rationality and is covered in depth in Chapter 16.

The notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents. Consider a clock. It can be thought of as just an inanimate object, or it can be thought of as a simple agent. As an agent, most clocks always do the right action: moving their hands (or displaying digits) in the proper fashion. Clocks are a kind of degenerate agent in that their percept sequence is empty; no matter what happens outside, the clock's action should be unaffected.

Well, this is not quite true. If the clock and its owner take a trip from California to Australia, the right thing for the clock to do would be to turn itself back six hours. We do not get upset at our clocks for failing to do this because we realize that they are acting rationally, given their lack of perceptual equipment.[3]

---

[3] One of the authors still gets a small thrill when his computer successfully resets itself at daylight savings time.

The ideal mapping from percept sequences to actions

Once we realize that an agent's behavior depends only on its percept sequence to date, then we can describe any particular agent by making a table of the action it takes in response to each possible percept sequence. (For most agents, this would be a very long list—infinite, in fact, unless we place a bound on the length of percept sequences we want to consider.) Such a list is called a **mapping** from percept sequences to actions. We can, in principle, find out which mapping correctly describes an agent by trying out all possible percept sequences and recording which actions the agent does in response. (If the agent uses some randomization in its computations, then we would have to try some percept sequences several times to get a good idea of the agent's average behavior.) And if mappings describe agents, then **ideal mappings** describe ideal agents. *Specifying which action an agent ought to take in response to any given percept sequence provides a design for an ideal agent.*

MAPPING

IDEAL MAPPINGS

This does not mean, of course, that we have to create an explicit table with an entry for every possible percept sequence. It is possible to define a specification of the mapping without exhaustively enumerating it. Consider a very simple agent: the square-root function on a calculator. The percept sequence for this agent is a sequence of keystrokes representing a number, and the action is to display a number on the display screen. The ideal mapping is that when the percept is a positive number x, the right action is to display a positive number z such that $z^2 \approx x$, accurate to, say, 15 decimal places. This specification of the ideal mapping does not require the designer to actually construct a table of square roots. Nor does the square-root function have to use a table to behave correctly: Figure 2.2 shows part of the ideal mapping and a simple program that implements the mapping using Newton's method.

The square-root example illustrates the relationship between the ideal mapping and an ideal agent design, for a very restricted task. Whereas the table is very large, the agent is a nice, compact program. It turns out that it is possible to design nice, compact agents that implement

Percept x	Action z
1.0	1.000000000000000
1.1	1.048808848170152
1.2	1.095445115010332
1.3	1.140175425099138
1.4	1.183215956619923
1.5	1.224744871391589
1.6	1.264911064067352
1.7	1.303840481040530
1.8	1.341640786499874
1.9	1.378404875209022
⋮	⋮

```
function SQRT($x$)
  z ← 1.0                /* initial guess */
  repeat until $|z^2 - x| < 10^{-15}$
      $z \leftarrow z - (z^2 - x)/(2z)$
  end
  return z
```

Figure 2.2 Part of the ideal mapping for the square-root problem (accurate to 15 digits), and a corresponding program that implements the ideal mapping.

the ideal mapping for much more general situations: agents that can solve a limitless variety of tasks in a limitless variety of environments. Before we discuss how to do this, we need to look at one more requirement that an intelligent agent ought to satisfy.

Autonomy

AUTONOMY

There is one more thing to deal with in the definition of an ideal rational agent: the "built-in knowledge" part. If the agent's actions are based completely on built-in knowledge, such that it need pay no attention to its percepts, then we say that the agent lacks **autonomy.** For example, if the clock manufacturer was prescient enough to know that the clock's owner would be going to Australia at some particular date, then a mechanism could be built in to adjust the hands automatically by six hours at just the right time. This would certainly be successful behavior, but the intelligence seems to belong to the clock's designer rather than to the clock itself.

An agent's behavior can be based on both its own experience and the built-in knowledge used in constructing the agent for the particular environment in which it operates. *A system is autonomous*[4] *to the extent that its behavior is determined by its own experience.* It would be too stringent, though, to require complete autonomy from the word go: when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance. So, just as evolution provides animals with enough built-in reflexes so that they can survive long enough to learn for themselves, it would be reasonable to provide an artificial intelligent agent with some initial knowledge as well as an ability to learn.

Autonomy not only fits in with our intuition, but it is an example of sound engineering practices. An agent that operates on the basis of built-in assumptions will only operate successfully when those assumptions hold, and thus lacks flexibility. Consider, for example, the lowly dung beetle. After digging its nest and laying its eggs, it fetches a ball of dung from a nearby heap to plug the entrance; if the ball of dung is removed from its grasp *en route,* the beetle continues on and pantomimes plugging the nest with the nonexistent dung ball, never noticing that it is missing. Evolution has built an assumption into the beetle's behavior, and when it is violated, unsuccessful behavior results. A truly autonomous intelligent agent should be able to operate successfully in a wide variety of environments, given sufficient time to adapt.

2.3 STRUCTURE OF INTELLIGENT AGENTS

AGENT PROGRAM

ARCHITECTURE

So far we have talked about agents by describing their *behavior*—the action that is performed after any given sequence of percepts. Now, we will have to bite the bullet and talk about how the insides work. The job of AI is to design the **agent program:** a function that implements the agent mapping from percepts to actions. We assume this program will run on some sort of computing device, which we will call the **architecture.** Obviously, the program we choose has to be one that the architecture will accept and run. The architecture might be a plain computer, or it might include special-purpose hardware for certain tasks, such as processing camera images or filtering audio input. It might also include software that provides a degree of insulation between the raw computer and the agent program, so that we can program at a higher level. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated. The relationship among agents, architectures, and programs can be summed up as follows:

---

[4] The word "autonomous" has also come to mean something like "not under the immediate control of a human," as in "autonomous land vehicle." We are using it in a stronger sense.

agent = architecture + program

Most of this book is about designing agent programs, although Chapters 24 and 25 deal directly with the architecture.

Before we design an agent program, we must have a pretty good idea of the possible percepts and actions, what goals or performance measure the agent is supposed to achieve, and what sort of environment it will operate in.[5] These come in a wide variety. Figure 2.3 shows the basic elements for a selection of agent types.

It may come as a surprise to some readers that we include in our list of agent types some programs that seem to operate in the entirely artificial environment defined by keyboard input and character output on a screen. "Surely," one might say, "this is not a real environment, is it?" In fact, what matters is not the distinction between "real" and "artificial" environments, but the complexity of the relationship among the behavior of the agent, the percept sequence generated by the environment, and the goals that the agent is supposed to achieve. Some "real" environments are actually quite simple. For example, a robot designed to inspect parts as they come by on a conveyer belt can make use of a number of simplifying assumptions: that the lighting is always just so, that the only thing on the conveyer belt will be parts of a certain kind, and that there are only two actions—accept the part or mark it as a reject.

SOFTWARE AGENTS

SOFTBOTS

In contrast, some **software agents** (or software robots or **softbots**) exist in rich, unlimited domains. Imagine a softbot designed to fly a flight simulator for a 747. The simulator is a very detailed, complex environment, and the software agent must choose from a wide variety of actions in real time. Or imagine a softbot designed to scan online news sources and show the interesting items to its customers. To do well, it will need some natural language processing abilities, it will need to learn what each customer is interested in, and it will need to dynamically change its plans when, for example, the connection for one news source crashes or a new one comes online.

Some environments blur the distinction between "real" and "artificial." In the ALIVE environment (Maes *et al.*, 1994), software agents are given as percepts a digitized camera image of a room where a human walks about. The agent processes the camera image and chooses an action. The environment also displays the camera image on a large display screen that the human can watch, and superimposes on the image a computer graphics rendering of the software agent. One such image is a cartoon dog, which has been programmed to move toward the human (unless he points to send the dog away) and to shake hands or jump up eagerly when the human makes certain gestures.

---

[5] For the acronymically minded, we call this the PAGE (Percepts, Actions, Goals, Environment) description. Note that the goals do *not* necessarily have to be represented within the agent; they simply describe the performance measure by which the agent design will be judged.

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Figure 2.3 Examples of agent types and their PAGE descriptions.

The most famous artificial environment is the Turing Test environment, in which the whole point is that real and artificial agents are on equal footing, but the environment is challenging enough that it is very difficult for a software agent to do as well as a human. Section 2.4 describes in more detail the factors that make some environments more demanding than others.

Agent programs

We will be building intelligent agents throughout the book. They will all have the same skeleton, namely, accepting percepts from an environment and generating actions. The early versions of agent programs will have a very simple form (Figure 2.4). Each will use some internal data structures that will be updated as new percepts arrive. These data structures are operated on by the agent's decision-making procedures to generate an action choice, which is then passed to the architecture to be executed.

There are two things to note about this skeleton program. First, even though we defined the agent mapping as a function from percept *sequences* to actions, the agent program receives only a single percept as its input. It is up to the agent to build up the percept sequence in memory, if it so desires. In some environments, it is possible to be quite successful without storing the percept sequence, and in complex domains, it is infeasible to store the complete sequence.

```
function SKELETON-AGENT(percept) returns action
   static: memory, the agent's memory of the world

   memory — UPDATE-MEMORY(memory, percept)
   action ← CHOOSE-BEST-ACTION(memory)
   memory — UPDATE-MEMORY(memory, action)
   return action
```

Figure 2.4 A skeleton agent. On each invocation, the agent's memory is updated to reflect the new percept, the best action is chosen, and the fact that the action was taken is also stored in memory. The memory persists from one invocation to the next.

Second, the goal or performance measure is *not* part of the skeleton program. This is because the performance measure is applied externally to judge the behavior of the agent, and it is often possible to achieve high performance without explicit knowledge of the performance measure (see, e.g., the square-root agent).

Why not just look up the answers?

Let us start with the simplest possible way we can think of to write the agent program—a lookup table. Figure 2.5 shows the agent program. It operates by keeping in memory its entire percept sequence, and using it to index into *table,* which contains the appropriate action for all possible percept sequences.

It is instructive to consider why this proposal is doomed to failure:

1. The table needed for something as simple as an agent that can only play chess would be about 35^{100} entries.
2. It would take quite a long time for the designer to build the table.
3. The agent has no autonomy at all, because the calculation of best actions is entirely built-in. So if the environment changed in some unexpected way, the agent would be lost.

```
function TABLE-DRIVEN-AGENT(percept) returns action
   static: percepts, a sequence, initially empty
           table, a table, indexed by percept sequences, initially fully specified

   append percept to the end of percepts
   action ← LOOKUP(percepts, table)
   return action
```

Figure 2.5 An agent based on a prespecified lookup table. It keeps track of the percept sequence and just looks up the best action.

4. Even if we gave the agent a learning mechanism as well, so that it could have a degree of autonomy, it would take forever to learn the right value for all the table entries.

Despite all this, TABLE-DRIVEN-AGENT *does* do what we want: it implements the desired agent mapping. It is not enough to say, "It can't be intelligent;" the point is to understand why an agent that *reasons* (as opposed to looking things up in a table) can do even better by avoiding the four drawbacks listed here.

An example

At this point, it will be helpful to consider a particular environment, so that our discussion can become more concrete. Mainly because of its familiarity, and because it involves a broad range of skills, we will look at the job of designing an automated taxi driver. We should point out, before the reader becomes alarmed, that such a system is currently somewhat beyond the capabilities of existing technology, although most of the components are available in some form.[6] The full driving task is extremely *open-ended*—there is no limit to the novel combinations of circumstances that can arise (which is another reason why we chose it as a focus for discussion).

We must first think about the percepts, actions, goals and environment for the taxi. They are summarized in Figure 2.6 and discussed in turn.

Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Figure 2.6 The taxi driver agent type.

The taxi will need to know where it is, what else is on the road, and how fast it is going. This information can be obtained from the **percepts** provided by one or more controllable TV cameras, the speedometer, and odometer. To control the vehicle properly, especially on curves, it should have an accelerometer; it will also need to know the mechanical state of the vehicle, so it will need the usual array of engine and electrical system sensors. It might have instruments that are not available to the average human driver: a satellite global positioning system (GPS) to give it accurate position information with respect to an electronic map; or infrared or sonar sensors to detect distances to other cars and obstacles. Finally, it will need a microphone or keyboard for the passengers to tell it their destination.

The actions available to a taxi driver will be more or less the same ones available to a human driver: control over the engine through the gas pedal and control over steering and braking. In addition, it will need output to a screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles.

---

[6] See page 26 for a description of an existing driving robot, or look at the conference proceedings on Intelligent Vehicle and Highway Systems (IVHS).

What **performance measure** would we like our automated driver to aspire to? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time and/or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so there will be trade-offs involved.

Finally, were this a real project, we would need to decide what kind of driving **environment** the taxi will face. Should it operate on local roads, or also on freeways? Will it be in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not? Will it always be driving on the right, or might we want it to be flexible enough to drive on the left in case we want to operate taxis in Britain or Japan? Obviously, the more restricted the environment, the easier the design problem.

Now we have to decide how to build a real program to implement the mapping from percepts to action. We will find that different aspects of driving suggest different types of agent program. We will consider four types of agent program:

- Simple reflex agents
- Agents that keep track of the world
- Goal-based agents
- Utility-based agents

Simple reflex agents

The option of constructing an explicit lookup table is out of the question. The visual input from a single camera comes in at the rate of 50 megabytes per second (25 frames per second, 1000 x 1000 pixels with 8 bits of color and 8 bits of intensity information). So the lookup table for an hour would be $2^{60 \times 60 \times 50M}$ entries.

However, we can summarize portions of the table by noting certain commonly occurring input/output associations. For example, if the car in front brakes, and its brake lights come on, then the driver should notice this and initiate braking. In other words, some processing is done on the visual input to establish the condition we call "The car in front is braking"; then this triggers some established connection in the agent program to the action "initiate braking". We call such a connection a **condition-action rule**[7] written as

CONDITION-ACTION RULE

if *car-in-front-is-braking* **then** *initiate-braking*

Humans also have many such connections, some of which are learned responses (as for driving) and some of which are innate reflexes (such as blinking when something approaches the eye). In the course of the book, we will see several different ways in which such connections can be learned and implemented.

Figure 2.7 gives the structure of a simple reflex agent in schematic form, showing how the condition-action rules allow the agent to make the connection from percept to action. (Do not worry if this seems trivial; it gets more interesting shortly.) We use rectangles to denote

---

[7] Also called **situation-action rules, productions, or if–then rules.** The last term is also used by some authors for logical implications, so we will avoid it altogether.

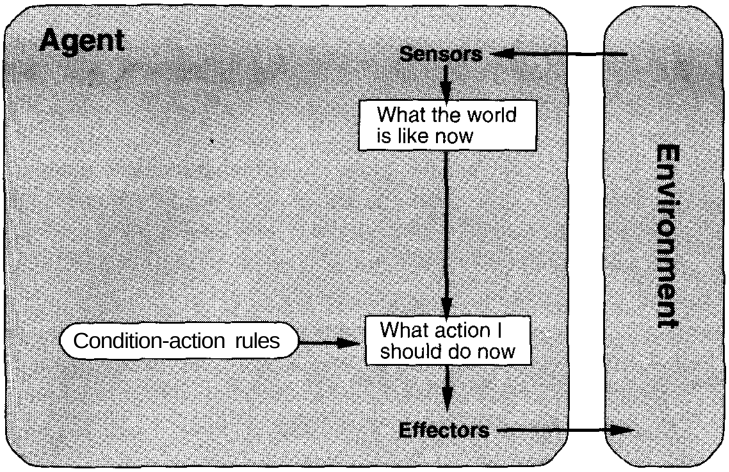

Figure 2.7 Schematic diagram of a simple reflex agent.

```
function SIMPLE-REFLEX-AGENT(percept) returns action
   static: rules, a set of condition-action rules

   state ← INTERPRET-INPUT(percept)
   rule ← RULE-MATCH(state, rules)
   action ← RULE-ACTION[rule]
   return action
```

Figure 2.8 A simple reflex agent. It works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule.

the current internal state of the agent's decision process, and ovals to represent the background information used in the process. The agent program, which is also very simple, is shown in Figure 2.8. The INTERPRET-INPUT function generates an abstracted description of the current state from the percept, and the RULE-MATCH function returns the first rule in the set of rules that matches the given state description. Although such agents can be implemented very efficiently (see Chapter 10), their range of applicability is very narrow, as we shall see.

Agents that keep track of the world

The simple reflex agent described before will work only if the correct decision can be made on the basis of the current percept. If the car in front is a recent model, and has the centrally mounted brake light now required in the United States, then it will be possible to tell if it is braking from a single image. Unfortunately, older models have different configurations of tail lights, brake lights, and turn-signal lights, and it is not always possible to tell if the car is braking. Thus, even for the simple braking rule, our driver will have to maintain some sort of **internal state** in order to choose an action. Here, the internal state is not too extensive—it just needs the previous frame from the camera to detect when two red lights at the edge of the vehicle go on or off simultaneously.

INTERNAL STATE

Consider the following more obvious case: from time to time, the driver looks in the rear-view mirror to check on the locations of nearby vehicles. When the driver is not looking in the mirror, the vehicles in the next lane are invisible (i.e., the states in which they are present and absent are indistinguishable); but in order to decide on a lane-change maneuver, the driver needs to know whether or not they are there.

The problem illustrated by this example arises because the sensors do not provide access to the complete state of the world. In such cases, the agent may need to maintain some internal state information in order to distinguish between world states that generate the same perceptual input but nonetheless are significantly different. Here, "significantly different" means that different actions are appropriate in the two states.

Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program. First, we need some information about how the world evolves independently of the agent—for example, that an overtaking car generally will be closer behind than it was a moment ago. Second, we need some information about how the agent's own actions affect the world—for example, that when the agent changes lanes to the right, there is a gap (at least temporarily) in the lane it was in before, or that after driving for five minutes northbound on the freeway one is usually about five miles north of where one was five minutes ago.

Figure 2.9 gives the structure of the reflex agent, showing how the current percept is combined with the old internal state to generate the updated description of the current state. The agent program is shown in Figure 2.10. The interesting part is the function UPDATE-STATE, which is responsible for creating the new internal state description. As well as interpreting the new percept in the light of existing knowledge about the state, it uses information about how the world evolves to keep track of the unseen parts of the world, and also must know about what the agent's actions do to the state of the world. Detailed examples appear in Chapters 7 and 17.

Goal-based agents

Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, right, or go straight on. The right decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information, which describes situations that are desirable—for example, being at the passenger's destination. The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose actions that achieve the goal. Sometimes this will be simple, when goal satisfaction results immediately from a single action; sometimes, it will be more tricky, when the agent has to consider long sequences of twists and turns to find a way to achieve the goal. **Search** (Chapters 3 to 5) and **planning** (Chapters 11 to 13) are the subfields of **AI** devoted to finding action sequences that do achieve the agent's goals.

GOAL

SEARCH

PLANNING

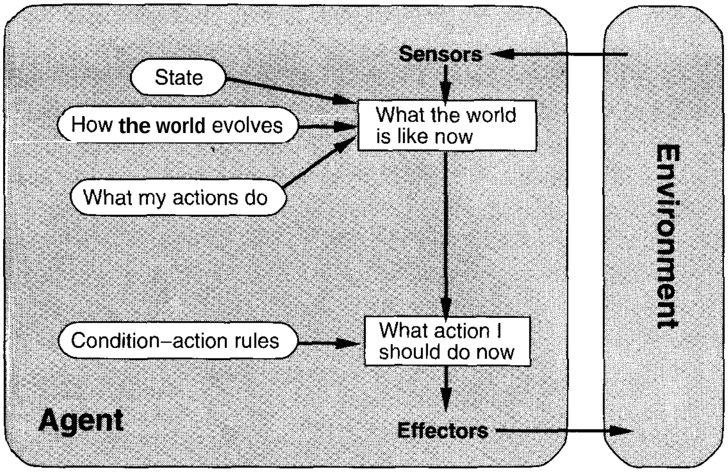

Figure 2.9 A reflex agent with internal state.

```
**function** REFLEX-AGENT-WITH-STATE(*percept*) **returns** *action*
    **static:** *state*, a description of the current world state
            *rules*, a set of condition-action rules

    state ← UPDATE-STATE(state, percept)
    rule — RULE-MATCH(state, rules)
    action — RULE-ACTION[rule]
    state ← UPDATE-STATE(state, action)
    return action
```

Figure 2.10 A reflex agent with internal state. It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule.

Notice that decision-making of this kind is fundamentally different from the condition-action rules described earlier, in that it involves consideration of the future—both "What will happen if I do such-and-such?" and "Will that make me happy?" In the reflex agent designs, this information is not explicitly used, because the designer has precomputed the correct action for various cases. The reflex agent brakes when it sees brake lights. A goal-based agent, in principle, could reason that if the car in front has its brake lights on, it will slow down. From the way the world usually evolves, the only action that will achieve the goal of not hitting other cars is to brake. Although the goal-based agent appears less efficient, it is far more flexible. If it starts to rain, the agent can update its knowledge of how effectively its brakes will operate; this will automatically cause all of the relevant behaviors to be altered to suit the new conditions. For the reflex agent, on the other hand, we would have to rewrite a large number of condition-action rules. Of course, the goal-based agent is also more flexible with respect to reaching different destinations. Simply by specifying a new destination, we can get the goal-based agent to come up with a new behavior. The reflex agent's rules for when to turn and when to go straight will only work for a single destination; they must all be replaced to go somewhere new.

Figure 2.11 shows the goal-based agent's structure. Chapter 13 contains detailed agent programs for goal-based agents.

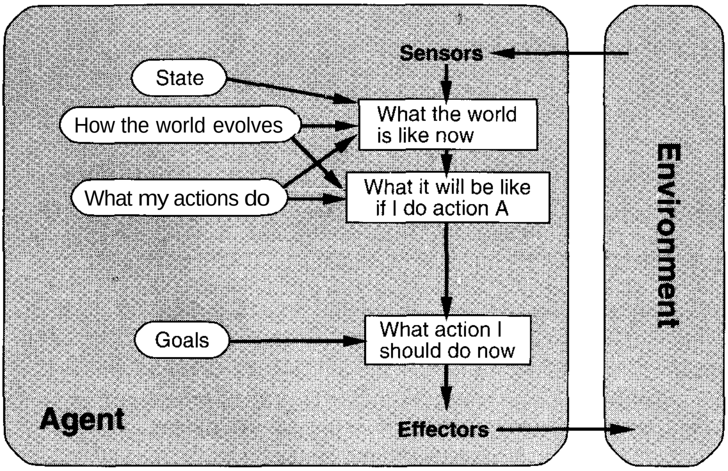
Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Effectors
Environment

Figure 2.11 An agent with explicit goals.

Utility-based agents

Goals alone are not really enough to generate high-quality behavior. For example, there are many action sequences that will get the taxi to its destination, thereby achieving the goal, but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude distinction between "happy" and "unhappy" states, whereas a more general performance measure should allow a comparison of different world states (or sequences of states) according to exactly how happy they would make the agent if they could be achieved. Because "happy" does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher **utility** for the agent.[8]

UTILITY

Utility is therefore a function that maps a state[9] onto a real number, which describes the associated degree of happiness. A complete specification of the utility function allows rational decisions in two kinds of cases where goals have trouble. First, when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate trade-off. Second, when there are several goals that the agent can aim for, none of which can *be* achieved with certainty, utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.

---

[8] The word "utility" here refers to "the quality of being useful," not to the electric company or water works.

[9] Or sequence of states, if we are measuring the utility of an agent over the long run.

In Chapter 16, we show that any rational agent can be described as possessing a utility function. An agent that possesses an *explicit* utility function therefore can make rational decisions, but may have to compare the utilities achieved by different courses of actions. Goals, although cruder, enable the agent to pick an action right away if it satisfies the goal. In some cases, moreover, a utility function can be translated into a set of goals, such that the decisions made by a goal-based agent using those goals are identical to those made by the utility-based agent.

The overall utility-based agent structure appears in Figure 2.12. Actual utility-based agent programs appeal in Chapter 5, where we examine game-playing programs that must make fine distinctions amoag various board positions; and in Chapter 17, where we tackle the general problem of designing decision-making agents.

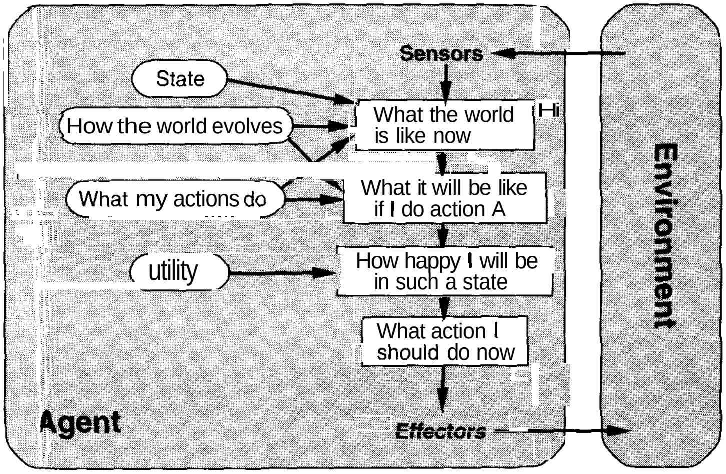
Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
utility
How happy I will be in such a state
What action I should do now
Agent
Effectors
Environment

Figure 2.12 A complete utility-based agent.

2.4 ENVIRONMENTS

In this section and in the exercises at the end of the chapter, you will see how to couple an agent to an environment. Section 2.3 introduced several different kinds of agents and environments. In all cases, however, the nature of the connection between them is the same: actions are done by the agent on the environment, which in turn provides percepts to the agent. First, we will describe the different types of environments and how they affect the design of agents. Then we will describe environment programs that can be used as testbeds for agent programs.

Properties of environments

Environments come in several flavors. The principal distinctions to be made are as follows:

ACCESSIBLE

0 **Accessible** vs. **inaccessible.**
If an agent's sensory apparatus gives it access to the complete state of the environment, then we say that the environment is accessible to that agent. An environment is effectively accessible if the sensors detect all aspects that are relevant to the choice of action. An accessible environment is convenient because the agent need not maintain any internal state to keep track of the world.

DETERMINISTIC

0 **Deterministic** vs. **nondeterministic**.
If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic. In principle, an agent need not worry about uncertainty in an accessible, deterministic environment. If the environment is inaccessible, however, then it may *appear* to be nondeterministic. This is particularly true if the environment is complex, making it hard to keep track of all the inaccessible aspects. Thus, it is often better to think of an environment as deterministic or nondeterministic *from the point of view of the agent.*

EPISODIC

0 **Episodic** vs. **nonepisodic.**
In an episodic environment, the agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

STATIC

0 **Static** vs. **dynamic.**
If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. If the environment does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**.

SEMIDYNAMIC

DISCRETE

0 **Discrete** vs. **continuous.**
If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Chess is discrete—there are a fixed number of possible moves on each turn. Taxi driving is continuous—the speed and location of the taxi and the other vehicles sweep through a range of continuous values.[10]

We will see that different environment types require somewhat different agent programs to deal with them effectively. It will turn out, as you might expect, that the hardest case is *inaccessible, nonepisodic, dynamic,* and *continuous.* It also turns out that most real situations are so complex that whether they are *really* deterministic is a moot point; for practical purposes, they must be treated as nondeterministic.

---

[10] At a fine enough level of granularity, even the taxi driving environment is discrete, because the camera image is digitized to yield discrete pixel values. But any sensible agent program would have to abstract above this level, up to a level of granularity that is continuous.

Figure 2.13 lists the properties of a number of familiar environments. Note that the answers can change depending on how you conceptualize the environments and agents. For example, poker is deterministic if the agent can keep track of the order of cards in the deck, but it is nondeterministic if it cannot. Also, many environments are episodic at higher levels than the agent's individual actions. For example, a chess tournament consists of a sequence of games; each game is an episode, because (by and large) the contribution of the moves in one game to the agent's overall performance is not affected by the moves in its next game. On the other hand, moves within a single game certainly interact, so the agent needs to look ahead several moves.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Figure 2.13 Examples of environments and their characteristics.

Environment programs

The generic environment program in Figure 2.14 illustrates the basic relationship between agents and environments. In this book, we will find it convenient for many of the examples and exercises to use an environment simulator that follows this program structure. The simulator takes one or more agents as input and arranges to repeatedly give each agent the right percepts and receive back an action. The simulator then updates the environment based on the actions, and possibly other dynamic processes in the environment that are not considered to be agents (rain, for example). The environment is therefore defined by the initial state and the update function. Of course, an agent that works in a simulator ought also to work in a real environment that provides the same kinds of percepts and accepts the same kinds of actions.

The RUN-ENVIRONMENT procedure correctly exercises the agents in an environment. For some kinds of agents, such as those that engage in natural language dialogue, it may be sufficient simply to observe their behavior. To get more detailed information about agent performance, we insert some performance measurement code. The function RUN-EVAL-ENVIRONMENT, shown in Figure 2.15, does this; it applies a performance measure to each agent and returns a list of the resulting scores. The *scores* variable keeps track of each agent's score.

In general, the performance measure can depend on the entire sequence of environment states generated during the operation of the program. Usually, however, the performance measure

```
procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
  inputs: state, the initial state of the environment
          UPDATE-FN, function to modify the environment
          agents, a set of agents
          termination, a predicate to test when we are done

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent,state)
     end
     for each agent in agents do
        ACTION[agent] — PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
  until termination(state)
```

Figure 2.14 The basic environment simulator program. It gives each agent its percept, gets an action from each agent, and then updates the environment.

```
function RUN-EVAL-ENVIRONMENT(state, UPDATE-FN, agents,
                              termination, PERFORMANCE-FN) returns scores
  local variables: scores, a vector the same size as agents, all 0

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent, state)
     end
     for each agent in agents do
        ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
     scores ← PERFORMANCE-FN(scores, agents, state)
  until termination(state)
  return scores                                         I * change * /
```

Figure 2.15 An environment simulator program that keeps track of the performance measure for each agent.

works by a simple accumulation using either summation, averaging, or taking a maximum. For example, if the performance measure for a vacuum-cleaning agent is the total amount of dirt cleaned in a shift, *scores* will just keep track of how much dirt has been cleaned up so far.

RUN-EVAL-ENVIRONMENT returns the performance measure for a a single environment, defined by a single initial state and a particular update function. Usually, an agent is designed to

ENVIRONMENT CLASS

work in an **environment class,** a whole set of different environments. For example, we design a chess program to play against any of a wide collection of human and machine opponents. If we designed it for a single opponent, we might be able to take advantage of specific weaknesses in that opponent, but that would not give us a good program for general play. Strictly speaking, in order to measure the performance of an agent, we need to have an environment generator that selects particular environments (with certain likelihoods) in which to run the agent. We are then interested in the agent's average performance over the environment class. This is fairly straightforward to implement for a simulated environment, and Exercises 2.5 to 2.11 take you through the entire development of an environment and the associated measurement process.

A possible confusion arises between the state variable in the environment simulator and the state variable in the agent itself (see REFLEX-AGENT-WITH-STATE). As a programmer implementing both the environment simulator and the agent, it is tempting to allow the agent to peek at the environment simulator's state variable. This temptation must be resisted at all costs! The agent's version of the state must be constructed from its percepts alone, without access to the complete state information.

2.5 SUMMARY

This chapter has been something of a whirlwind tour of AI, which we have conceived of as the science of agent design. The major points to recall are as follows:

- **An agent** is something that perceives and acts in an environment. We split an agent into an architecture and an agent program.
- **An ideal agent** is one that always takes the action that is expected to maximize its performance measure, given the percept sequence it has seen so far.
- An agent is **autonomous** to the extent that its action choices depend on its own experience, rather than on knowledge of the environment that has been built-in by the designer.
- **An agent program** maps from a percept to an action, while updating an internal state.
- There exists a variety of basic agent program designs, depending on the kind of information made explicit and used in the decision process. The designs vary in efficiency, compactness, and flexibility. The appropriate design of the agent program depends on the percepts, actions, goals, and environment.
- **Reflex agents** respond immediately to percepts, **goal-based agents** act so that they will achieve their goal(s), and **utility-based agents** try to maximize their own "happiness."
- The process of making decisions by reasoning with knowledge is central to AI and to successful agent design. This means that representing knowledge is important.
- Some environments are more demanding than others. Environments that are inaccessible, nondeterministic, nonepisodic, dynamic, and continuous are the most challenging.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The analysis of rational agency as a mapping from percept sequences to actions probably stems ultimately from the effort to identify rational behavior in the realm of economics and other forms of reasoning under uncertainty (covered in later chapters) and from the efforts of psychological behaviorists such as Skinner (1953) to reduce the psychology of organisms strictly to input/output or stimulus/response mappings. The advance from behaviorism to functionalism in psychology, which was at least partly driven by the application of the computer metaphor to agents (Putnam, 1960; Lewis, 1966), introduced the internal state of the agent into the picture. The philosopher Daniel Dennett (1969; 1978b) helped to synthesize these viewpoints into a coherent "intentional stance" toward agents. A high-level, abstract perspective on agency is also taken within the world of AI in (McCarthy and Hayes, 1969). Jon Doyle (1983) proposed that rational agent design is the core of AI, and would remain as its mission while other topics in AI would spin off to form new disciplines. Horvitz *et al.* (1988) specifically suggest the use of rationality conceived as the maximization of expected utility as a basis for AI.

The AI researcher and Nobel-prize-winning economist Herb Simon drew a clear distinction between rationality under resource limitations (procedural rationality) and rationality as making the objectively rational choice (substantive rationality) (Simon, 1958). Cherniak (1986) explores the minimal level of rationality needed to qualify an entity as an agent. Russell and Wefald (1991) deal explicitly with the possibility of using a variety of agent architectures. *Dung Beetle Ecology* (Hanski and Cambefort, 1991) provides a wealth of interesting information on the behavior of dung beetles.

EXERCISES

2.1 What is the difference between a performance measure and a utility function?

2.2 For each of the environments in Figure 2.3, determine what type of agent architecture is most appropriate (table lookup, simple reflex, goal-based or utility-based).

2.3 Choose a domain that you are familiar with, and write a PAGE description of an agent for the environment. Characterize the environment as being accessible, deterministic, episodic, static, and continuous or not. What agent architecture is best for this domain?

2.4 While driving, which is the best policy?

a. Always put your directional blinker on before turning,
b. Never use your blinker,
c. Look in your mirrors and use your blinker only if you observe a car that can observe you?

What kind of reasoning did you need to do to arrive at this policy (logical, goal-based, or utility-based)? What kind of agent design is necessary to carry out the policy (reflex, goal-based, or utility-based)?

The following exercises all concern the implementation of an environment and set of agents in the vacuum-cleaner world.

2.5 Implement a performance-measuring environment simulator for the vacuum-cleaner world. This world can be described as follows:

◇ **Percepts:** Each vacuum-cleaner agent gets a three-element percept vector on each turn. The first element, a touch sensor, should be a 1 if the machine has bumped into something and a 0 otherwise. The second comes from a photosensor under the machine, which emits a 1 if there is dirt there and a 0 otherwise. The third comes from an infrared sensor, which emits a 1 when the agent is in its home location, and a 0 otherwise.

0 Actions: There are five actions available: go forward, turn right by 90°, turn left by 90°, suck up dirt, and turn off.

◇ **Goals:** The goal for each agent is to clean up and go home. To be precise, the performance measure will be 100 points for each piece of dirt vacuumed up, minus 1 point for each action taken, and minus 1000 points if it is not in the home location when it turns itself off.

◇ **Environment:** The environment consists of a grid of squares. Some squares contain obstacles (walls and furniture) and other squares are open space. Some of the open squares contain dirt. Each "go forward" action moves one square unless there is an obstacle in that square, in which case the agent stays where it is, but the touch sensor goes on. A "suck up dirt" action always cleans up the dirt. A "turn off" command ends the simulation.

We can vary the complexity of the environment along three dimensions:

◇ **Room shape:** In the simplest case, the room is an n x n square, for some fixed n. We can make it more difficult by changing to a rectangular, L-shaped, or irregularly shaped room, or a series of rooms connected by corridors.

0 **Furniture:** Placing furniture in the room makes it more complex than an empty room. To the vacuum-cleaning agent, a piece of furniture cannot be distinguished from a wall by perception; both appear as a 1 on the touch sensor.

0 **Dirt placement:** In the simplest case, dirt is distributed uniformly around the room. But it is more realistic for the dirt to predominate in certain locations, such as along a heavily travelled path to the next room, or in front of the couch.

2.6 Implement a table-lookup agent for the special case of the vacuum-cleaner world consisting of a 2 x 2 grid of open squares, in which at most two squares will contain dirt. The agent starts in the upper left corner, facing to the right. Recall that a table-lookup agent consists of a table of actions indexed by a percept sequence. In this environment, the agent can always complete its task in nine or fewer actions (four moves, three turns, and two suck-ups), so the table only needs entries for percept sequences up to length nine. At each turn, there are eight possible percept vectors, so the table will be of size $8^9 = 134,217,728$. Fortunately, we can cut this down by realizing that the touch sensor and home sensor inputs are not needed; we can arrange so that the agent never bumps into a wall and knows when it has returned home. Then there are only two relevant percept vectors, ?0? and ?1?, and the size of the table is at most $2^9 = 512$. Run the environment simulator on the table-lookup agent in all possible worlds (how many are there?). Record its performance score for each world and its overall average score.

2.7 Implement an environment for a $n \times m$ rectangular room, where each square has a 5% chance of containing dirt, and n and m are chosen at random from the range 8 to 15, inclusive.

2.8 Design and implement a pure reflex agent for the environment of Exercise 2.7, ignoring the requirement of returning home, and measure its performance. Explain why it is impossible to have a reflex agent that returns home and shuts itself off. Speculate on what the best possible reflex agent could do. What prevents a reflex agent from doing very well?

2.9 Design and implement several agents with internal state. Measure their performance. How close do they come to the ideal agent for this environment?

2.10 Calculate the size of the table for a table-lookup agent in the domain of Exercise 2.7. Explain your calculation. You need not fill in the entries for the table.

2.11 Experiment with changing the shape and dirt placement of the room, and with adding furniture. Measure your agents in these new environments. Discuss how their performance might be improved to handle more complex geographies.