

# Искусственный интеллект

Современный подход

*Второе издание*

# Artificial Intelligence

## A Modern Approach

*Second edition*

Stuart J. Russel and Peter Norvig



Prentice Hall  
Upper Saddle River, New Jersey 07458

# Искусственный интеллект

## Современный подход

*Второе издание*

Стюарт Рассел, Питер Норвиг

Москва • Санкт-Петербург • Киев  
2007



ББК 32.973.26-018.2.75

P24

УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция *К.А. Птицына*

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:

[info@williamspublishing.com](mailto:info@williamspublishing.com), <http://www.williamspublishing.com>

115419, Москва, а/я 783; 03150, Киев, а/я 152

**Рассел, Стюарт, Норвиг, Питер.**

P24 Искусственный интеллект: современный подход, 2-е изд.. : Пер. с англ. — М. :

Издательский дом “Вильямс”, 2007. — 1408 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-0887-2 (рус.)

В книге представлены все современные достижения и изложены идеи, которые были сформулированы в исследованиях, проводившихся в течение последних пятидесяти лет, а также собраны на протяжении двух тысячелетий в областях знаний, ставших стимулом к развитию искусственного интеллекта как науки проектирования рациональных агентов. Теоретическое описание иллюстрируется многочисленными алгоритмами, реализации которых в виде готовых программ на нескольких языках программирования находятся на сопровождающем книгу Web-узле.

Книга предназначена для использования в базовом университетском курсе или в последовательности курсов по специальности. Применима в качестве основного справочника для аспирантов, специализирующихся в области искусственного интеллекта, а также будет небезинтересна профессионалам, желающим выйти за пределы избранной ими специальности. Благодаря кристальной ясности и наглядности изложения вполне может быть отнесена к лучшим образцам научно-популярной литературы.

**ББК 32.973.26-018.2.75**

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фоторепродукцию и запись на магнитный носитель, если на это нет письменного разрешения издательства Prentice Hall, Inc.

Authorized translation from the English language edition published by Prentice Hall, Copyright © 2003 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition was published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2007

ISBN 978-5-8459-0887-2 (рус.)

ISBN 0-13-790395-2 (англ.)

© Издательский дом “Вильямс”, 2007

© by Pearson Education, Inc., 2003

## **ОГЛАВЛЕНИЕ**

---

Предисловие	24
Об авторах	31
<b>ЧАСТЬ I. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ</b>	33
Глава 1. Введение	34
Глава 2. Интеллектуальные агенты	75
<b>ЧАСТЬ II. РЕШЕНИЕ ПРОБЛЕМ</b>	109
Глава 3. Решение проблем посредством поиска	110
Глава 4. Информированный поиск и исследование пространства состояний	153
Глава 5. Задачи удовлетворения ограничений	209
Глава 6. Поиск в условиях противодействия	240
<b>ЧАСТЬ III. ЗНАНИЯ И РАССУЖДЕНИЯ</b>	281
Глава 7. Логические агенты	282
Глава 8. Логика первого порядка	341
Глава 9. Логический вывод в логике первого порядка	380
Глава 10. Представление знаний	440
<b>ЧАСТЬ IV. ПЛАНИРОВАНИЕ</b>	511
Глава 11. Основы планирования	512
Глава 12. Планирование и осуществление действий в реальном мире	564
<b>ЧАСТЬ V. НЕОПРЕДЕЛЕННЫЕ ЗНАНИЯ И РАССУЖДЕНИЯ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ</b>	621
Глава 13. Неопределенность	622
Глава 14. Вероятностные рассуждения	660
Глава 15. Вероятностные рассуждения во времени	718
Глава 16. Принятие простых решений	778
Глава 17. Принятие сложных решений	815
<b>ЧАСТЬ VI. ОБУЧЕНИЕ</b>	863
Глава 18. Обучение на основе наблюдений	864
Глава 19. Применение знаний в обучении	902
Глава 20. Статистические методы обучения	945
Глава 21. Обучение с подкреплением	1010

<b>ЧАСТЬ VII. Общение, восприятие и осуществление действий</b>	1045
Глава 22. Общение	1046
Глава 23. Вероятностная обработка лингвистической информации	1102
Глава 24. Восприятие	1141
Глава 25. Робототехника	1188
 <b>ЧАСТЬ VIII. ЗАКЛЮЧЕНИЕ</b>	 1247
Глава 26. Философские основания	1248
Глава 27. Настоящее и будущее искусственного интеллекта	1277
Приложение А. Математические основы	1288
Приложение Б. Общие сведения о языках и алгоритмах, используемых в книге	1297
Литература	1302
Предметный указатель	1373

## СОДЕРЖАНИЕ

---

Предисловие	24
Краткий обзор книги	25
Отличия от первого издания	26
Как использовать эту книгу	27
Использование Web-узла	28
Благодарности	28
Об обложке	30
Об авторах	31
 ЧАСТЬ I. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ	33
 ГЛАВА 1. ВВЕДЕНИЕ	34
1.1. Общее определение искусственного интеллекта	34
Проверка того, способен ли компьютер действовать подобно человеку: подход, основанный на использовании теста Тьюринга	36
Как мыслить по-человечески: подход, основанный на когнитивном моделировании	37
Как мыслить рационально: подход, основанный на использовании "законов мышления"	38
Как мыслить рационально: подход, основанный на использовании рационального агента	39
1.2. Предыстория искусственного интеллекта	40
Философия (период с 428 года до н.э. по настоящее время)	40
Математика (период примерно с 800 года по настоящее время)	43
Экономика (период с 1776 года по настоящее время)	45
Неврология (период с 1861 года по настоящее время)	46
Психология (период с 1879 года по настоящее время)	49
Вычислительная техника (период с 1940 года по настоящее время)	51
Теория управления и кибернетика (период с 1948 года по настоящее время)	52
Лингвистика (период с 1957 года по настоящее время)	53
1.3. История искусственного интеллекта	54
Появление предпосылок искусственного интеллекта (период с 1943 года по 1955 год)	54
Рождение искусственного интеллекта (1956 год)	55
Ранний энтузиазм, большие ожидания (период с 1952 года по 1969 год)	56
Столкновение с реальностью (период с 1966 года по 1973 год)	60
Системы, основанные на знаниях: могут ли они стать ключом к успеху (период с 1969 года по 1979 год)	62

Превращение искусственного интеллекта в индустрию (период с 1980 года по настоящее время)	65
Возвращение к нейронным сетям (период с 1986 года по настоящее время)	65
Превращение искусственного интеллекта в науку (период с 1987 года по настоящее время)	66
Появление подхода, основанного на использовании интеллектуальных агентов (период с 1995 года по настоящее время)	68
1.4. Современное состояние разработок	69
1.5. Резюме	71
Библиографические и исторические заметки	72
Упражнения	73
 ГЛАВА 2. ИНТЕЛЛЕКТУАЛЬНЫЕ АГЕНТЫ	75
2.1. Агенты и варианты среды	75
2.2. Качественное поведение: концепция рациональности	78
Показатели производительности	78
Рациональность	79
Всезнание, обучение и автономность	80
2.3. Определение характера среды	82
Определение проблемной среды	83
Свойства проблемной среды	86
2.4. Структура агентов	90
Программы агентов	91
Простые рефлексные агенты	93
Рефлексные агенты, основанные на модели	96
Агенты, основанные на цели	97
Агенты, основанные на полезности	99
Обучающиеся агенты	100
2.5. Резюме	103
Библиографические и исторические заметки	104
Упражнения	106
 ЧАСТЬ II. РЕШЕНИЕ ПРОБЛЕМ	109
 ГЛАВА 3. РЕШЕНИЕ ПРОБЛЕМ ПОСРЕДСТВОМ ПОИСКА	110
3.1. Агенты, решающие задачи	110
Хорошо структурированные задачи и решения	113
Формулировка задачи	115
3.2. Примеры задач	116
Упрощенные задачи	116
Реальные задачи	120
3.3. Поиск решений	122
Измерение производительности решения задачи	126
3.4. Стратегии неинформированного поиска	127
Поиск в ширину	127

Поиск в глубину	130
Поиск с ограничением глубины	131
Поиск в глубину с итеративным углублением	133
Двунаправленный поиск	135
Сравнение стратегий неинформированного поиска	136
3.5. Предотвращение формирования повторяющихся состояний	136
3.6. Поиск с частичной информацией	139
Проблемы отсутствия датчиков	140
Проблемы непредвиденных ситуаций	142
3.7. Резюме	144
Библиографические и исторические заметки	145
Упражнения	147
 ГЛАВА 4. ИНФОРМИРОВАННЫЙ ПОИСК И ИССЛЕДОВАНИЕ ПРОСТРАНСТВА СОСТОЯНИЙ	
	153
4.1. Стратегии информированного (эвристического) поиска	154
Жадный поиск по первому наилучшему совпадению	155
Поиск A*: минимизация суммарной оценки стоимости решения	157
Эвристический поиск с ограничением объема памяти	163
Обучение лучшим способам поиска	166
4.2. Эвристические функции	167
Зависимость производительности поиска от точности эвристической функции	168
Составление допустимых эвристических функций	170
Изучение эвристических функций на основе опыта	173
4.3. Алгоритмы локального поиска и задачи оптимизации	174
Поиск с восхождением к вершине	175
Поиск с эмуляцией отжига	180
Локальный лучевой поиск	181
Генетические алгоритмы	182
4.4. Локальный поиск в непрерывных пространствах	187
4.5. Поисковые агенты, действующие в оперативном режиме, и неизвестные варианты среды	189
Задачи поиска в оперативном режиме	190
Агенты, выполняющие поиск в оперативном режиме	193
Локальный поиск в оперативном режиме	194
Обучение в ходе поиска в оперативном режиме	197
4.6. Резюме	198
Библиографические и исторические заметки	199
Упражнения	204
 ГЛАВА 5. ЗАДАЧИ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ	
	209
5.1. Задачи удовлетворения ограничений	210
5.2. Применение поиска с возвратами для решения задач CSP	214
Упорядочение переменных и значений	217
Распространение информации с помощью ограничений	219

Интеллектуальный поиск с возвратами: поиск в обратном направлении	224
5.3. Применение локального поиска для решения задач удовлетворения ограничений	226
5.4. Структура задач	228
5.5. Резюме	233
Библиографические и исторические заметки	233
Упражнения	236
 ГЛАВА 6. ПОИСК В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ	240
6.1. Игры	240
6.2. Принятие оптимальных решений в играх	242
Оптимальные стратегии	242
Минимаксный алгоритм	245
Оптимальные решения в играх с несколькими игроками	246
6.3. Альфа-бета-отсечение	247
6.4. Неидеальные решения, принимаемые в реальном времени	252
Функции оценки	252
Прекращение поиска	254
6.5. Игры, которые включают элемент случайности	257
Оценка позиции в играх с узлами жеребьевки	260
Сложность оценки ожидаемых минимаксных значений	261
Карточные игры	262
6.6. Современные игровые программы	264
6.7. Обсуждение изложенных сведений	268
6.8. Резюме	270
Библиографические и исторические заметки	271
Упражнения	276
 ЧАСТЬ III. ЗНАНИЯ И РАССУЖДЕНИЯ	281
 ГЛАВА 7. ЛОГИЧЕСКИЕ АГЕНТЫ	282
7.1. Агенты, основанные на знаниях	284
7.2. Мир вампуса	286
7.3. Логика	290
7.4. Пропозициональная логика: очень простая логика	294
Синтаксис	295
Семантика	296
Простая база знаний	299
Логический вывод	300
Эквивалентность, допустимость и выполнимость	301
7.5. Шаблоны формирования рассуждений в пропозициональной логике	303
Резолюция	306
Прямой и обратный логический вывод	311
7.6. Эффективный пропозициональный логический вывод	316
Полный алгоритм поиска с возвратами	316

Алгоритмы локального поиска	318
Трудные задачи определения выполнимости	320
7.7. Агенты, основанные на пропозициональной логике	322
Поиск ям и вампусов с помощью логического вывода	322
Слежение за местонахождением и ориентацией	324
Агенты на основе логических схем	325
Сопоставление двух описанных типов агентов	330
7.8. Резюме	332
Библиографические и исторические заметки	333
Упражнения	337
 ГЛАВА 8. ЛОГИКА ПЕРВОГО ПОРЯДКА	 341
8.1. Дополнительные сведения о представлении	341
8.2. Синтаксис и семантика логики первого порядка	347
Модели для логики первого порядка	347
Символы и интерпретации	349
Термы	351
Атомарные высказывания	351
Сложные высказывания	352
Кванторы	352
Равенство	357
8.3. Использование логики первого порядка	357
Утверждения и запросы в логике первого порядка	358
Проблемная область родства	358
Числа, множества и списки	361
Мир вампуша	363
8.4. Инженерия знаний с применением логики первого порядка	366
Процесс инженерии знаний	367
Проблемная область электронных схем	369
8.5. Резюме	374
Библиографические и исторические заметки	374
Упражнения	376
 ГЛАВА 9. ЛОГИЧЕСКИЙ ВЫВОД В ЛОГИКЕ ПЕРВОГО ПОРЯДКА	 380
9.1. Сравнение методов логического вывода в пропозициональной логике	380
и логике первого порядка	381
Правила логического вывода для кванторов	381
Приведение к пропозициональному логическому выводу	382
9.2. Унификация и поднятие	384
Правило вывода в логике первого порядка	384
Унификация	386
Хранение и выборка	388
9.3. Прямой логический вывод	390
Определенные выражения в логике первого порядка	390
Простой алгоритм прямого логического вывода	392
Эффективный прямой логический вывод	394

9.4. Обратный логический вывод	399
Алгоритм обратного логического вывода	399
Логическое программирование	401
Эффективная реализация логических программ	403
Избыточный логический вывод и бесконечные циклы	406
Логическое программирование в ограничениях	408
9.5. Резолюция	409
Конъюнктивная нормальная форма для логики первого порядка	410
Правило логического вывода с помощью резолюции	412
Примеры доказательств	413
Полнота резолюции	416
Учет отношения равенства	420
Стратегии резолюции	421
Средства автоматического доказательства теорем	423
9.6. Резюме	428
Библиографические и исторические заметки	429
Упражнения	435
 ГЛАВА 10. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ	440
10.1. Онтологическая инженерия	440
10.2. Категории и объекты	443
Физическая композиция	445
Меры	448
Вещества и объекты	449
10.3. Действия, ситуации и события	451
Онтология ситуационного исчисления	451
Описание действий в ситуационном исчислении	453
Решение проблемы представительного окружения	455
Решение проблемы выводимого окружения	457
Исчисление времени и событий	459
Обобщенные события	460
Процессы	462
Интервалы	464
Флюентные высказывания и объекты	465
10.4. Мыслительные события и мыслимые объекты	466
Формальная теория убеждений	467
Знания и убеждения	469
Знания, время и действия	470
10.5. Мир покупок в Internet	471
Сравнение коммерческих предложений	476
10.6. Системы формирования рассуждений о категориях	477
Семантические сети	478
Описательные логики	482
10.7. Формирование рассуждений с использованием информации, заданной по умолчанию	483
Открытые и закрытые миры	484

Отрицание как недостижение цели и устойчивая семантика модели	486
Логика косвенного описания и логика умолчания	488
10.8. Системы поддержки истинности	491
10.9. Резюме	494
Библиографические и исторические заметки	495
Упражнения	503
<b>ЧАСТЬ IV. ПЛАНИРОВАНИЕ</b>	<b>511</b>
<b>ГЛАВА 11. ОСНОВЫ ПЛАНИРОВАНИЯ</b>	<b>512</b>
11.1. Задача планирования	513
Язык задач планирования	514
Выразительность и расширения языка	516
Пример: воздушный грузовой транспорт	518
Пример: задача с запасным колесом	519
Пример: мир блоков	520
11.2. Планирование с помощью поиска в пространстве состояний	521
Прямой поиск в пространстве состояний	522
Обратный поиск в пространстве состояний	523
Эвристики для поиска в пространстве состояний	525
11.3. Планирование с частичным упорядочением	527
Пример планирования с частичным упорядочением	532
Планирование с частичным упорядочением и несвязанными переменными	534
Эвристики для планирования с частичным упорядочением	535
11.4. Графы планирования	536
Применение графов планирования для получения эвристической оценки	539
Алгоритм Graphplan	541
Завершение работы алгоритма Graphplan	544
11.5. Планирование с помощью пропозициональной логики	545
Описание задач планирования в пропозициональной логике	546
Сложности, связанные с использованием пропозициональных кодировок	549
11.6. Анализ различных подходов к планированию	551
11.7. Резюме	553
Библиографические и исторические заметки	554
Упражнения	558
<b>ГЛАВА 12. ПЛАНИРОВАНИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ В РЕАЛЬНОМ МИРЕ</b>	<b>564</b>
12.1. Время, расписания и ресурсы	564
Составление расписаний с ресурсными ограничениями	567
12.2. Планирование иерархической сети задач	570
Представление декомпозиций действий	572
Модификация планировщика для его использования в сочетании с декомпозициями	574

Обсуждение вопроса	577
12.3. Планирование и осуществление действий в недетерминированных проблемных областях	580
12.4. Условное планирование	584
Условное планирование в полностью наблюдаемых вариантах среды	584
Условное планирование в частично наблюдаемых вариантах среды	589
12.5. Контроль выполнения и перепланирование	594
12.6. Непрерывное планирование	600
12.7. Мультиагентное планирование	605
Кооперация: совместные цели и планы	605
Многотельное планирование	606
Механизмы координации	608
Конкуренция	610
12.8. Резюме	611
Библиографические и исторические заметки	612
Упражнения	616
<b>ЧАСТЬ V. НЕОПРЕДЕЛЕННЫЕ ЗНАНИЯ И РАССУЖДЕНИЯ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ</b>	<b>621</b>
<b>ГЛАВА 13. НЕОПРЕДЕЛЕННОСТЬ</b>	<b>622</b>
13.1. Действия в условиях неопределенности	622
Учет наличия неопределенных знаний	623
Неопределенность и рациональные решения	626
Проект агента, действующего в соответствии с теорией решений	627
13.2. Основная вероятностная система обозначений	628
Высказывания	628
Атомарные события	629
Априорная вероятность	630
Условная вероятность	632
13.3. Аксиомы вероятностей	635
Использование аксиом вероятностей	635
Теоретическое обоснование аксиом вероятностей	636
13.4. Логический вывод с использованием полных совместных распределений	638
13.5. Независимость	642
13.6. Правило Байеса и его использование	644
Применение правила Байеса: простой случай	644
Использование правила Байеса: комбинирование свидетельств	646
13.7. Еще одно возвращение в мир вампуша	648
13.8. Резюме	652
Библиографические и исторические заметки	653
Упражнения	656
<b>ГЛАВА 14. ВЕРОЯТНОСТНЫЕ РАССУЖДЕНИЯ</b>	<b>660</b>
14.1. Представление знаний в неопределенной проблемной области	660

14.2. Семантика байесовских сетей	664
Представление полного совместного распределения	664
Отношения условной независимости в байесовских сетях	669
14.3. Эффективное представление распределений условных вероятностей	669
14.4. Точный вероятностный вывод в байесовских сетях	675
Вероятностный вывод с помощью перебора	676
Алгоритм устранения переменной	678
Сложность точного вероятностного вывода	681
Алгоритмы кластеризации	682
14.5. Приближенный вероятностный вывод в байесовских сетях	683
Методы непосредственной выборки	684
Вероятностный вывод по методу моделирования цепи Маркова	690
14.6. Распространение вероятностных методов на представления в логике первого порядка	694
14.7. Другие подходы к формированию рассуждений в условиях неопределенности	699
Методы на основе правил для формирования рассуждений в условиях неопределенности	700
Представление незнания: теория Демпстера–Шефера	703
Представление неосведомленности: нечеткие множества и нечеткая логика	704
14.8. Резюме	706
Библиографические и исторические заметки	707
Упражнения	712
 ГЛАВА 15. ВЕРОЯТНОСТНЫЕ РАССУЖДЕНИЯ ВО ВРЕМЕНИ	718
15.1. Время и неопределенность	719
Состояния и наблюдения	719
Стационарные процессы и марковское предположение	720
15.2. Вероятностный вывод во временных моделях	724
Фильтрация и предсказание	725
Сглаживание	728
Поиск наиболее вероятной последовательности	731
15.3. Скрытые марковские модели	734
Упрощенные матричные алгоритмы	734
15.4. Фильтры Калмана	737
Обновление гауссовых распределений	738
Простой одномерный пример	739
Общий случай	743
Области применения калмановской фильтрации	744
15.5. Динамические байесовские сети	746
Процедура создания сетей DBN	747
Точный вероятностный вывод в сетях DBN	752
Приближенный вероятностный вывод в сетях DBN	754
15.6. Распознавание речи	758
Звуки речи	760

Слова	763
Предложения	765
Разработка устройства распознавания речи	769
15.7. Резюме	770
Библиографические и исторические заметки	771
Упражнения	774
 ГЛАВА 16. ПРИНЯТИЕ ПРОСТЫХ РЕШЕНИЙ	778
16.1. Совместный учет убеждений и желаний в условиях неопределенности	778
16.2. Основы теории полезности	780
Ограничения, налагаемые на рациональные предпочтения	781
В начале была Полезность	783
16.3. Функции полезности	784
Полезность денег	784
Шкалы полезности и оценка полезности	788
16.4. Многоатрибутные функции полезности	790
Доминирование	790
Структура предпочтений и многоатрибутная полезность	793
16.5. Сети принятия решений	795
Способы представления задачи принятия решений с помощью сети принятия решений	795
Вычисления с помощью сетей принятия решений	798
16.6. Стоимость информации	798
Простой пример	799
Общая формула	800
Свойства показателей стоимости информации	802
Реализация агента, действующего на основе сбора информации	802
16.7. Экспертные системы, основанные на использовании теории принятия решений	803
16.8. Резюме	807
Библиографические и исторические заметки	808
Упражнения	810
 ГЛАВА 17. ПРИНЯТИЕ СЛОЖНЫХ РЕШЕНИЙ	815
17.1. Задачи последовательного принятия решений	816
Пример	816
Оптимальность в задачах последовательного принятия решений	819
17.2. Итерация по значениям	822
Полезности состояний	823
Алгоритм итерации по значениям	824
Сходимость итерации по значениям	826
17.3. Итерация по стратегиям	829
17.4. Марковские процессы принятия решений в частично наблюдаемых вариантах среды	831
17.5. Агенты, действующие на основе теории решений	836
17.6. Принятие решений при наличии нескольких агентов: теория игр	839

17.7. Проектирование механизма	851
17.8. Резюме	855
Библиографические и исторические заметки	856
Упражнения	859
<b>ЧАСТЬ VI. ОБУЧЕНИЕ</b>	<b>863</b>
ГЛАВА 18. ОБУЧЕНИЕ НА ОСНОВЕ НАБЛЮДЕНИЙ	864
18.1. Формы обучения	864
18.2. Индуктивное обучение	867
18.3. Формирование деревьев решений на основе обучения	870
Деревья решений, рассматриваемые как производительные элементы	870
Выразительность деревьев решений	872
Индуктивный вывод деревьев решений на основе примеров	873
Выбор проверок атрибутов	877
Оценка производительности обучающего алгоритма	879
Шум и чрезмерно тщательная подгонка	880
Расширение области применения деревьев решений	883
18.4. Обучение ансамбля	884
18.5. Принципы функционирования алгоритмов обучения: теория вычислительного обучения	889
Оценка количества необходимых примеров	890
Обучение списков решений	892
Обсуждение полученных результатов	894
18.6. Резюме	895
Библиографические и исторические заметки	896
Упражнения	899
<b>ГЛАВА 19. ПРИМЕНЕНИЕ ЗНАНИЙ В ОБУЧЕНИИ</b>	<b>902</b>
19.1. Логическая формулировка задачи обучения	902
Примеры и гипотезы	903
Поиск текущей наилучшей гипотезы	905
Поиск на основе оценки наименьшего вклада	908
19.2. Применение знаний в обучении	913
Некоторые простые примеры	914
Некоторые общие схемы	915
19.3. Обучение на основе объяснения	917
Извлечение общих правил из примеров	919
Повышение эффективности правила	921
19.4. Обучение с использованием информации о релевантности	923
Определение пространства гипотез	923
Обучение и использование информации о релевантности	924
19.5. Индуктивное логическое программирование	927
Практический пример	928
Нисходящие методы индуктивного обучения	931

Индуктивное обучение с помощью обратной дедукции	934
Совершение открытий с помощью индуктивного логического программирования	937
19.6. Резюме	939
Библиографические и исторические заметки	940
Упражнения	943
 ГЛАВА 20. СТАТИСТИЧЕСКИЕ МЕТОДЫ ОБУЧЕНИЯ	945
20.1. Статистическое обучение	946
20.2. Обучение с помощью полных данных	950
Обучение параметрам с помощью метода максимального правдоподобия: дискретные модели	950
Наивные байесовские модели	953
Обучение параметрам с максимальным правдоподобием: непрерывные модели	954
Обучение байесовским параметрам	956
Определение путем обучения структур байесовских сетей	959
20.3. Обучение с помощью скрытых переменных: алгоритм EM	961
Неконтролируемая кластеризация: определение в процессе обучения смешанных гауссовых распределений	962
Обучение байесовских сетей со скрытыми переменными	966
Обучение скрытых марковских моделей	969
Общая форма алгоритма EM	970
Определение с помощью обучения структур байесовских сетей со скрытыми переменными	971
20.4. Обучение на основе экземпляра	972
Модели ближайшего соседа	973
Ядерные модели	975
20.5. Нейронные сети	976
Элементы в нейронных сетях	977
Структуры сетей	979
Однослойные нейронные сети с прямым распространением (персептроны)	980
Многослойные нейронные сети с прямым распространением	985
Определение в процессе обучения структур нейронных сетей	990
20.6. Ядерные машины	991
20.7. Практический пример: распознавание рукописных цифр	995
20.8. Резюме	998
Библиографические и исторические заметки	1000
Упражнения	1005
 ГЛАВА 21. ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ	1010
21.1. Введение	1010
21.2. Пассивное обучение с подкреплением	1012
Непосредственная оценка полезности	1014
Адаптивное динамическое программирование	1015

Обучение с учетом временной разницы	1016
21.3. Активное обучение с подкреплением	1020
Исследование среды	1021
Определение функции “действие–стоимость” с помощью обучения	1025
21.4. Обобщение в обучении с подкреплением	1027
Приложения методов обучения к ведению игр	1031
Применение к управлению роботами	1032
21.5. Поиск стратегии	1033
21.6. Резюме	1037
Библиографические и исторические заметки	1039
Упражнения	1042
 ЧАСТЬ VII. ОБЩЕНИЕ, ВОСПРИЯТИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ	1045
 ГЛАВА 22. ОБЩЕНИЕ	1046
22.1. Общение как действие	1047
Основные понятия языка	1048
Составные этапы общения	1050
22.2. Формальная грамматика для подмножества английского языка	1054
Словарь языка $\mathcal{E}_0$	1054
Грамматика языка $\mathcal{E}_0$	1055
22.3. Синтаксический анализ (синтаксический разбор)	1056
Эффективный синтаксический анализ	1058
22.4. Расширенные грамматики	1065
Субкатегоризация глагола	1068
Порождающая мощь расширенных грамматик	1071
22.5. Семантическая интерпретация	1071
Семантика небольшой части английского языка	1072
Время события и времена глаголов	1074
Введение кванторов	1075
Прагматическая интерпретация	1078
Применение грамматик DCG для производства языковых конструкций	1079
22.6. Неоднозначность и устранение неоднозначности	1080
Устранение неоднозначности	1083
22.7. Понимание речи	1085
Разрешение ссылок	1085
Структура связной речи	1087
22.8. Индуктивный вывод грамматики	1089
22.9. Резюме	1092
Библиографические и исторические заметки	1093
Упражнения	1097
 ГЛАВА 23. ВЕРОЯТНОСТНАЯ ОБРАБОТКА ЛИНГВИСТИЧЕСКОЙ ИНФОРМАЦИИ	1102
23.1. Вероятностные языковые модели	1103
Вероятностные контекстно-свободные грамматики	1106

Определение с помощью обучения вероятностей для грамматики PCFG	1108
Определение с помощью обучения структуры правил для грамматики PCFG	1110
<b>23.2. Информационный поиск</b>	1110
Сравнительный анализ систем информационного поиска	1114
Совершенствование информационного поиска	1115
Способы представления результирующих наборов	1117
Создание систем информационного поиска	1119
<b>23.3. Извлечение информации</b>	1121
<b>23.4. Машинный перевод</b>	1124
Системы машинного перевода	1127
Статистический машинный перевод	1127
Определение с помощью обучения вероятностей для машинного перевода	1132
<b>23.5. Резюме</b>	1134
Библиографические и исторические заметки	1135
Упражнения	1138
 ГЛАВА 24. ВОСПРИЯТИЕ	1141
<b>24.1. Введение</b>	1141
<b>24.2. Формирование изображения</b>	1143
Получение изображения без линз — камера-обскура	1144
Системы линз	1145
Свет: фотометрия формирования изображения	1146
Цвет — спектрофотометрия формирования изображения	1147
<b>24.3. Операции, выполняемые на первом этапе обработки изображения</b>	1148
Обнаружение краев	1150
Сегментация изображения	1153
<b>24.4. Извлечение трехмерной информации</b>	1154
Движение	1156
Бинокулярные стереоданные	1158
Градиенты текстуры	1161
Затенение	1162
Контуры	1164
<b>24.5. Распознавание объектов</b>	1168
Распознавание с учетом яркости	1171
Распознавание с учетом характеристик	1172
Оценка позы	1175
<b>24.6. Использование системы машинного зрения для манипулирования и передвижения</b>	1177
<b>24.7. Резюме</b>	1180
Библиографические и исторические заметки	1181
Упражнения	1184
 ГЛАВА 25. РОБОТОТЕХНИКА	1188
<b>25.1. Введение</b>	1188

25.2. Аппаратное обеспечение роботов	1190
Датчики	1190
Исполнительные механизмы	1192
25.3. Восприятие, осуществляемое роботами	1195
Локализация	1197
Составление карты	1203
Другие типы восприятия	1206
25.4. Планирование движений	1207
Пространство конфигураций	1207
Методы декомпозиции ячеек	1210
Методы скелетирования	1214
25.5. Планирование движений в условиях неопределенности	1215
Надежные методы	1217
25.6. Осуществление движений	1220
Динамика и управление	1220
Управление на основе поля потенциалов	1223
Реактивное управление	1225
25.7. Архитектуры робототехнического программного обеспечения	1227
Обобщаящая архитектура	1227
Трехуровневая архитектура	1229
Робототехнические языки программирования	1230
25.8. Прикладные области	1231
25.9. Резюме	1235
Библиографические и исторические заметки	1237
Упражнения	1241
 ЧАСТЬ VIII. ЗАКЛЮЧЕНИЕ	1247
 ГЛАВА 26. ФИЛОСОФСКИЕ ОСНОВАНИЯ	1248
26.1. Слабый искусственный интеллект: могут ли машины действовать интеллигентально?	1249
Довод, исходящий из неспособности	1250
Возражения, основанные на принципах математики	1251
Довод, исходящий из неформализуемости	1253
26.2. Сильный искусственный интеллект: могут ли машины по-настоящему мыслить?	1255
Проблема разума и тела	1258
Эксперимент “мозг в колбе”	1260
Эксперимент с протезом мозга	1261
Китайская комната	1263
26.3. Этические и моральные последствия разработки искусственного интеллекта	1266
26.4. Резюме	1271
Библиографические и исторические заметки	1272
Упражнения	1275

ГЛАВА 27. НАСТОЯЩЕЕ И БУДУЩЕЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА	1277
27.1. Компоненты агента	1278
27.2. Архитектуры агентов	1281
27.3. Оценка правильности выбранного направления	1283
27.4. Перспективы развития искусственного интеллекта	1285
 ПРИЛОЖЕНИЕ А. МАТЕМАТИЧЕСКИЕ ОСНОВЫ	1288
A.1. Анализ сложности и система обозначений O()	1288
Асимптотический анализ	1288
Изначально сложные и недетерминированные полиномиальные задачи	1290
A.2. Векторы, матрицы и линейная алгебра	1291
A.3. Распределения вероятностей	1293
Библиографические и исторические заметки	1295
 ПРИЛОЖЕНИЕ Б. ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКАХ И АЛГОРИТМАХ, ИСПОЛЬЗУЕМЫХ В КНИГЕ	1297
Б.1. Определение языков с помощью формы Бэкуса–Наура	1297
Б.2. Описание алгоритмов с помощью псевдокода	1298
Б.3. Оперативная помощь	1299
 ЛИТЕРАТУРА	1302
 ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	1373

*Посвящается Лой, Гордону и Люси*

С.Дж. Рассел

*Посвящается Крису, Изабелле и Джсульетте*

П. Норвиг

## ПРЕДИСЛОВИЕ

---

Искусственный интеллект (ИИ) — широкая область знаний; именно поэтому данная книга имеет такой большой объем. Авторы попытались достаточно полно описать теоретические основы искусственного интеллекта, включая математическую логику, теорию вероятностей и теорию непрерывных функций, раскрыть суть таких понятий, как восприятие, рассуждение, обучение и действие, а также описать все технические средства, созданные в рамках этого научного направления, начиная с микроэлектронных устройств и заканчивая межпланетными автоматическими зондами. Большой объем данной книги обусловлен также тем, что авторы стремились представить достигнутые результаты достаточно глубоко, хотя в основной части каждой главы они старались охватить только самые важные идеи, касающиеся рассматриваемой темы. Указания, позволяющие получить более полные сведения, приведены в библиографических заметках в конце каждой главы.

Эта книга имеет подзаголовок “Современный подход”. С помощью этого довольно-таки малосодержательного названия авторы хотели подчеркнуть, что пытались представить в ней в рамках единого способа изложения все современные достижения в области искусственного интеллекта, а не описать каждое отдельное его направление в его собственном историческом контексте. Авторы приносят свои извинения представителям тех направлений, которые стали выглядеть не столь значимыми, как они заслуживают, лишь из-за того, что для их описания принят такой подход.

Главной объединяющей темой этой книги является идея *интеллектуального агента*. Авторы определяют *искусственный интеллект* как науку об агентах, которые получают результаты актов восприятия из своей среды и выполняют действия, причем каждый такой агент реализует функцию, которая отображает последовательности актов восприятия в действия. В данной книге рассматриваются различные способы представления этих функций, в частности производственные системы, реактивные агенты, условные планировщики в реальном масштабе времени, нейронные сети и системы, действующие на основе теории решений. Авторы трактуют роль *обучения* как распространения сферы деятельности проектировщика на неизвестную среду и показывают, какие ограничения налагает указанный подход к обучению на проект агента, способствуя применению явного представления знаний и таких же способов формирования рассуждений. Кроме того, авторы рассматривают робототехнику и системы технического зрения не как независимо определяемые научные направления, а как области знаний, позволяющие обеспечить более успешное достижение целей, стоящих перед агентами, и подчеркивают важность учета того, в какой среде агент решает поставленные перед ним задачи, при определении соответствующего проекта агента.

Основная цель авторов состояла в том, чтобы изложить идеи, которые были сформулированы в исследованиях по искусственному интеллекту, проводившихся в течение последних пятидесяти лет, а также собраны на протяжении последних двух тысячелетий в тех областях знаний, которые стали стимулом к развитию искусственного интеллекта. Мы старались избегать чрезмерного формализма при изложении этих идей, сохраняя при этом необходимую точность. При любой возможности мы приводили алгоритмы на псевдокоде, чтобы конкретизировать излагаемые идеи;

краткие сведения о применяемом нами псевдокоде содержатся в приложении Б. Реализации этих алгоритмов на нескольких языках программирования можно найти на сопровождающем Web-узле книги ([aima.cs.berkeley.edu](http://aima.cs.berkeley.edu)).

Настоящая книга прежде всего предназначена для использования в базовом университетском курсе или в последовательности курсов. Она может также использоваться в курсе по специальности (возможно, с добавлением материала из некоторых основных источников, предложенных в библиографических заметках). Кроме того, данная книга характеризуется всесторонним охватом тематики и большим количеством подробных алгоритмов, поэтому применима в качестве основного справочника для аспирантов, специализирующихся в области искусственного интеллекта, а также будет небезинтересна профессионалам, желающим выйти за пределы избранной ими специальности. При этом единственным требованием является знакомство с основными понятиями информатики (алгоритмы, структуры данных, классы сложности) на уровне студента-второкурсника. Для понимания материала по нейронным сетям и подробного ознакомления со сведениями о статистическом обучении полезно освоить исчисление на уровне студента первого курса. Часть необходимых сведений из области математики приведена в приложении А.

## Краткий обзор книги

Данная книга разделена на восемь частей. В части I, “Искусственный интеллект”, предлагается общий обзор тематики искусственного интеллекта, базирующейся на идее интеллектуального агента — системы, которая способна принять решение о том, что делать, а затем выполнить это решение. В части II, “Решение проблем”, изложение сосредоточено на методах принятия решений по выбору оптимальных действий в тех условиях, когда необходимо продумывать наперед несколько этапов, например при поиске маршрута проезда через всю страну или при игре в шахматы. В части III, “Знания и рассуждения”, обсуждаются способы представления знаний о мире — как он функционирует, каковы его основные особенности в настоящее время и к чему могут привести те или иные действия, а также способы формирования логических рассуждений на основе этих знаний. В части IV, “Планирование”, описывается, как использовать эти способы формирования рассуждений для принятия решений по выбору дальнейших действий, особенно при составлении планов. Часть V, “Неопределенные знания и рассуждения в условиях неопределенности”, аналогична частям III и IV, но в ней изложение в основном сосредоточивается на способах формирования рассуждений и принятия решений в условиях неопределенности знаний о мире, с чем обычно приходится сталкиваться, например, в системах медицинской диагностики и лечения.

Части II–V, вместе взятые, содержат описание тех компонентов интеллектуального агента, которые отвечают за выработку решений. В части VI, “Обучение”, описаны методы выработки знаний, необходимых для этих компонентов, которые обеспечивают принятие решений. В части VII, “Общение, восприятие и осуществление действий”, описаны способы, с помощью которых интеллектуальные агенты могут получать результаты восприятия из своей среды, чтобы узнать, что в ней происходит, либо с помощью систем технического зрения, осязания, слуха, либо на основе понимания языка, а также способы, с помощью которых интеллектуальные агенты могут претворять свои планы в реальные действия, такие как выполнение движений

рбота или произнесение фрагментов речи на естественном языке. Наконец, в части VIII, “Заключение”, анализируется прошлое и будущее искусственного интеллекта и рассматриваются философские и этические последствия его развития.

## Отличия от первого издания

Со времени публикации первого издания этой книги в 1995 году в искусственном интеллекте многое изменилось, поэтому внесены значительные изменения и в саму книгу. Каждая глава была в значительной степени переработана, чтобы в ней можно было отразить новейшие достижения в рассматриваемой области, дать иное толкование старым работам с той точки зрения, которая более согласована с новыми результатами, а также улучшить качество изложения рассматриваемых идей в соответствии с принципами педагогики. Активных пользователей методов искусственного интеллекта должно порадовать то, что представленные в настоящем издании методы стали намного более эффективными по сравнению с теми, которые описывались в издании 1995 года; например, алгоритмы планирования, которые рассматривались в первом издании, позволяли формировать планы, состоящие всего лишь из нескольких шагов, тогда как масштабы применения алгоритмов, описанные в настоящем издании, увеличились до десятков тысяч шагов. Подобные усовершенствования, измеряемые несколькими порядками величин, достигнуты и в областях вероятностного вывода и обработки лингвистической информации, а также в других вспомогательных областях. Наиболее существенные изменения, внесенные во второе издание, описаны ниже.

- В части I изложены факты, которые свидетельствуют о признании исторического вклада в развитие искусственного интеллекта со стороны теории управления, теории игр, экономики и неврологии. Это позволяет создать основу для более целостного описания идей, заимствованных из этих научных областей, в последующих главах.
- В части II описаны алгоритмы оперативного поиска и введена новая глава по удовлетворению ограничений, которая позволяет установить естественную связь между вычислительными методами и приведенными в данной книге материалами по логике.
- Теперь в части III пропозициональная логика, которая в первом издании была рекомендована читателям как промежуточная ступенька на пути к логике первого порядка, рассматривается как полезный сам по себе язык представления, для которого предусмотрены быстродействующие алгоритмы логического вывода и эффективные проекты агентов на основе схемы. Главы по логике первого порядка были реорганизованы для более наглядного изложения материала, а в качестве примера проблемной области приведено описание процесса осуществления покупок в Internet.
- В части IV приведены сведения о более новых методах планирования, таких как Graphplan и планирование на основе выполнимости. Кроме того, увеличен объем изложения, касающегося составления расписаний, условного планирования, иерархического планирования и мультиагентского планирования.
- В часть V включен дополнительный материал по байесовским сетям, в котором описаны новые алгоритмы, в частности алгоритмы устранения перемен-

ных и алгоритмы Монте-Карло на основе марковской цепи, а также введена новая глава по формированию неопределенных рассуждений с учетом времени и созданию покрытий скрытых марковских моделей, а также по применению фильтров Калмана и динамических байесовских сетей. Описание марковских процессов принятия решений стало еще более глубоким; введены новые разделы по теории игр и проектированию механизма.

- В части VI связаны воедино все результаты, достигнутые в области статистического и символического обучения, а также обучения нейронных сетей; кроме того, введены разделы, содержащие сведения об увеличении производительности алгоритмов, алгоритме EM, обучении на основе экземпляра и о ядерных методах (о машинах поддерживающих векторов).
- В части VII к общему объему материала об обработке лингвистической информации добавлены разделы, касающиеся обработки речи и индуктивного вывода грамматики, а также глава по вероятностным языковым моделям, с учетом того, что область применения этих сведений должны стать информационный поиск и машинный перевод. В ходе изложения вопросов робототехники подчеркнута необходимость применения методов обработки неопределенных сенсорных данных, а в главе по системам технического зрения приведены уточненные сведения по распознаванию объектов.
- В части VIII предусмотрен дополнительный раздел, касающийся этических последствий развития искусственного интеллекта.

## Как использовать эту книгу

Книга состоит из 27 глав, причем для изучения каждой из них требуется примерно недельный объем лекций. Таким образом, для учебной проработки всей книги требуется последовательность курсов лекций, рассчитанная на два семестра. Еще один вариант состоит в том, что может быть составлен выборочный курс, удовлетворяющий интересы преподавателя и студента. Благодаря тому что в ней охвачена широкая тематика, эта книга может использоваться в качестве основы для многих курсов, начиная с коротких, вводных циклов лекций для начинающих и заканчивая специализированными курсами с углубленным изучением избранной темы для студентов последних лет обучения. На Web-узле, находящемся по адресу [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu), приведены программы курсов лекций, проводимых более чем в 600 университетах и колледжах, в основу которых было положено первое издание настоящей книги, а также даны рекомендации, позволяющие читателю найти программу курсов лекций, в наибольшей степени соответствующую его потребностям.

Книга включает 385 упражнений. Упражнения, требующие существенного объема программирования, отмечены значком в виде клавиатуры (). Проще всего эти упражнения можно выполнить, воспользовавшись архивом кода, который находится по адресу [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu). Некоторые из упражнений настолько велики, что их можно рассматривать как проекты с заданными сроками. Многие упражнения требуют проведения определенных исследований с помощью доступной литературы; они отмечены значком в виде книги (). Важные примечания отмечены значком в виде “указующего перста” () и выделены курсивным шрифтом. В книгу

включен обширный предметный указатель, состоящий из нескольких тысяч элементов, который поможет читателю найти нужную тему. Кроме того, значком с изображением руки, держащей карандаш (), и полужирным шрифтом отмечаются все новые термины, где впервые приведено их определение.

## Использование Web-узла

На Web-узле [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu) приведено следующее:

- реализации алгоритмов, описанных в книге, на нескольких языках программирования;
- список более чем 600 учебных заведений, в которых используется данная книга, сопровождающийся многочисленными ссылками на материалы курсов, доступные в оперативном режиме;
- аннотированный список более чем 800 ссылок на Web-узлы с полезными сведениями по искусственному интеллекту;
- списки дополнительных материалов и ссылок, относящихся к каждой главе;
- инструкции с описанием того, как присоединяться к дискуссионной группе, посвященной данной книге;
- инструкции с описанием того, как обратиться к авторам, чтобы передать им свои вопросы или комментарии;
- инструкции с описанием того, как сообщить об ошибках, обнаруженных в книге;
- копии рисунков из оригинала книги, а также слайды и другие материалы для преподавателей.

## Благодарности

Основная часть главы 24 (по системам технического зрения) написана Джитендой Маликом (Jitendra Malik). Глава 25 (по робототехнике) в основном написана Себастьяном Траном (Sebastian Thrun) для настоящего издания и Джоном Кэнни (John Canny) для первого издания. Дуг Эдвардс (Doug Edwards) провел исследование, на основании которого написаны исторические заметки для первого издания. Тим Хуанг (Tim Huang), Марк Паскин (Mark Paskin) и Синтия Бруинс (Cynthia Bruyns) оказали помощь при оформлении диаграмм и алгоритмов. Аллан Апт (Alan Apt), Сондра Чавес (Sondra Chavez), Тони Холм (Toni Holm), Джейк Вард (Jake Warde), Ирвин Закер (Irwin Zucker) и Камилла Трантакост (Camille Trentacoste), сотрудники издательства Prentice Hall, приложили большие усилия, чтобы помочь нам соблюсти намеченный график подготовки книги, и внесли много полезных предложений по оформлению и содержанию книги.

Стюарт хотел бы поблагодарить своих родителей за их постоянную помощь и поддержку, а также свою жену, Лой Шефлотт (Loy Shefrott), за ее бесконечное терпение и безграничную мудрость. Он надеется, что скоро эту книгу прочитают Гордон и Люси. Исключительно полезной для него была работа с RUGS (Russell's Unusual Group of Students — необыкновенная группа студентов Рассела).

Питер хотел бы поблагодарить своих родителей, Торстена и Герду, за то, что они очень помогли ему на первых порах, и свою жену Крис, детей и друзей за то, что

подбадривали его и терпели его отсутствие в течение тех долгих часов, когда он писал эту книгу, и тех еще более долгих часов, когда он снова ее переписывал.

Мы очень обязаны библиотекарям, работающим в университете г. Беркли, Стэнфордском университете, Массачусетском технологическом институте и агентству NASA, а также разработчикам узлов CiteSeer и Google, которые внесли революционные изменения в сам способ проведения исследований.

Мы буквально не с состояния выразить свою признательность всем тем, кто использовал данную книгу и внес свои предложения, но хотели бы поблагодарить за особо полезные комментарии следующих: Кшиштофа Апта (Krzysztof Apt), Эллери Эзиела (Ellery Aziel), Джефа Ван Баалена (Jeff Van Baalen), Брайена Бейкера (Brian Baker), Дона Баркера (Don Barker), Тони Баррета (Tony Barrett), Джеймса Ньютона Баса (James Newton Bass), Дона Била (Don Beal), Говарда Бека (Howard Beck), Вольфганга Бибеля (Wolfgang Bibel), Джона Биндера (John Binder), Лэрри Букмана (Larry Bookman), Дэвида Р. Боксолла (David R. Boxall), Герхарда Бревку (Gerhard Brewka), Селмера Бринсфорда (Selmer Bringsjord), Карла Бродли (Carla Brodley), Криса Брауна (Chris Brown), Вильгельма Бургера (Wilhelm Burger), Лорен Берка (Lauren Burka), Жоао Кашпоро (Joao Cachopo), Меррея Кэмбелла (Murray Campbell), Нормана Карвера (Norman Carver), Эммануеля Кастро (Emmanuel Castro), Анила Чакраварти (Anil Chakravarthy), Дэна Чизарика (Dan Chisarick), Роберто Сиполлу (Roberto Cipolla), Дэвида Коэна (David Cohen), Джеймса Коулмэна (James Coleman), Джюли Энн Компарини (Julie Ann Comparini), Гэри Котрелла (Gary Cottrell), Эрнеста Дэвиса (Ernest Davis), Рину Дехтер (Rina Dechter), Тома Дицтерика (Tom Dietterich), Чака Дийера (Chuck Dyer), Барбару Энгельхардт (Barbara Engelhardt), Дуга Эдвардса (Doug Edwards), Кутлухана Эрола (Kutluhan Erol), Орена Этциони (Oren Etzioni), Хану Филип (Hana Filip), Дугласа Фишера (Douglas Fisher), Джейфри Форбса (Jeffrey Forbes), Кена Форда (Ken Ford), Джона Фослера (John Fosler), Алекса Франца (Alex Franz), Боба Фатрелла (Bob Futrelle), Марека Галецки (Marek Galecki), Штефана Гербердинга (Stefan Gerberding), Стюарта Джилла (Stuart Gill), Сабину Глеснер (Sabine Glesner), Сета Голуба (Seth Golub), Гошту Гранье (Gosta Grahne), Расса Грейнера (Russ Greiner), Эрика Гримсона (Eric Grimson), Барбару Грош (Barbara Grosz), Лэрри Холла (Larry Hall), Стива Хэнкса (Steve Hanks), Отара Ханссона (Othar Hansson), Эрнста Хайнца (Ernst Heinz), Джима Эндлера (Jim Hendler), Кристофа Херманна (Christoph Herrmann), Вазанта Хонавара (Vasant Honavar), Тима Хуанга (Tim Huang), Сета Хатчinsona (Seth Hutchinson), Джуста Джейкоба (Joost Jacob), Магнуса Йоханссона (Magnus Johansson), Дэна Джурафски (Dan Jurafsky), Лесли Кэлблинга (Leslie Kaelbling), Кейдзи Канадзawa (Keiji Kanazawa), Сурекха Касибхатла (Surekha Kasibhatla), Саймона Казифа (Simon Kasif), Генри Каутца (Henry Kautz), Гернота Кершбаумера (Gernot Kerschbaumer), Ричарда Кирби (Richard Kirby), Кевина Найта (Kevin Knight), Свена Кёнига (Sven Koenig), Дафну Коллер (Daphne Koller), Рича Корфа (Rich Korf), Джеймса Керина (James Kurien), Джона Лафферти (John Lafferty), Гаса Ларссона (Gus Larsson), Джона Лашаро (John Lazzaro), Джона Лебланка (Jon LeBlanc), Джейсона Литермана (Jason Leatherman), Фрэнка Ли (Frank Lee), Эдварда Лима (Edward Lim), Пьера Луво (Pierre Louveaux), Дона Лавленда (Don Loveland), Сридхара Махадевана (Sridhar Mahadevan), Джима Мартина (Jim Martin), Энди Мейера (Andy Mayer), Дэвида Мак-Грейна (David McGrane), Джей Мендельсон (Jay Mendelsohn), Брайена Милча (Brian Milch), Стива Майнтона (Steve Minton), Вибу Миттала (Vibhu Mittal), Леору Моргенстern (Leora Morgenstern), Стивена Магглтона (Stephen Muggleton), Кевина Мэрфи (Kevin Murphy), Рона Мьюзика (Ron Musick), Санга Миаэнга (Sung Myaeng), Ли Нэйша (Lee Naish), Панду Найака (Pandu

Nayak), Бернхарда Небеля (Bernhard Nebel), Стюарта Нельсона (Stuart Nelson), Шуанлонга Нгуэн (XuanLong Nguyen), Иллаха Нурбакш (Illah Nourbakhsh), Стива Омохандро (Steve Omohundro), Дэвида Пейджа (David Page), Дэвида Палмера (David Palmer), Дэвида Паркса (David Parkes), Рона Парра (Ron Parr), Марка Паскина (Mark Paskin), Тони Пассера (Tony Passera), Майкла Паззани (Michael Pazzani), Вима Пейлса (Wim Pijls), Иру Пол (Ira Pohl), Марту Поллак (Martha Pollack), Дэвида Пула (David Poole), Брюса Портера (Bruce Porter), Малкома Прадхана (Malcolm Pradhan), Билла Прингла (Bill Pringle), Лоррэйн Прайор (Lorraine Prior), Грэга Прована (Greg Provan), Уильяма Рапапорта (William Rapaport), Филипа Ресника (Philip Resnik), Франческу Росси (Francesca Rossi), Джонатана Шеффера (Jonathan Schaeffer), Ричарда Шерла (Richard Scherl), Ларса Шустера (Lars Schuster), Сохеиль Шамс (Soheil Shams), Стюарта Шапиро (Stuart Shapiro), Джюд Шавлик (Jude Shavlik), Сатиндера Сингха (Satinder Singh), Дэниела Слитора (Daniel Sleator), Дэвида Смита (David Smith), Брайена Сой (Bryan So), Роберта Спрула (Robert Sproull), Линн Стейн (Lynn Stein), Лэрри Стивенса (Larry Stephens), Андреаса Штолке (Andreas Stolcke), Пола Страйдинга (Paul Stradling), Девику Субраманиан (Devika Subramanian), Рича Саттона (Rich Sutton), Джонатана Тэша (Jonathan Tash), Остина Тэйта (Austin Tate), Майкла Тильтера (Michael Thielscher), Уильяма Томпсона (William Thompson), Себастьяна Трана (Sebastian Thrun), Эрика Тидеманна (Eric Tiedemann), Марка Торранса (Mark Torrance), Рэндалла Уфама (Randall Upham), Пола Утгоффа (Paul Utgoff), Питера ван Бека (Peter van Beek), Хала Вариана (Hal Varian), Сунила Вемури (Sunil Vemuri), Джима Уолдо (Jim Waldo), Бонни Веббер (Bonnie Webber), Дэна Вэлда (Dan Weld), Майкла Веллмана (Michael Wellman), Майкла Дина Уайта (Michael Dean White), Камины Уайтхауса (Kamini Whitehouse), Брайена Уильямса (Brian Williams), Дэвида Уолфа (David Wolfe), Билла Вудса (Bill Woods), Олдена Райта (Alden Wright), Ричарда Йэна (Richard Yen), Вэйшионг Джанг (Weixiong Zhang), Шломо Зильберштейна (Shlomo Zilberstein), а также анонимных рецензентов, привлеченных издательством Prentice Hall.

## Об обложке

Изображение на обложке было спроектировано авторами и выполнено Лайзой Мэри Сарденья (Lisa Marie Sardegna) и Мэриэнн Симmons (Maryann Simmons) с использованием программ SGI Inventor™ и Adobe Photoshop™. На обложке показаны перечисленные ниже предметы, иллюстрирующие историю искусственного интеллекта.

1. Алгоритм планирования Аристотеля из книги *De Motu Animalium* (ок. 400 до н.э.).
2. Генератор понятий Раймунда Луллия из книги *Ars Magna* (ок. 1300).
3. Разностная машина Чарльза Бэббиджа, прототип первого универсального компьютера (1848).
4. Система обозначений Готтлоба Фреге для логики первого порядка (1789).
5. Диаграммы Льюиса Кэрролла для формирования логических рассуждений (1886).
6. Система обозначений вероятностной сети Сьюэлла Райта (1921).
7. Алан Тьюринг (1912–1954).
8. Робот Shakey (1969–1973).
9. Современная диагностическая экспертная система (1993).

## ОБ АВТОРАХ

---

Стюарт Рассел родился в 1962 году в г. Портсмут, Англия. Он получил степень бакалавра искусств по физике с наградами первой степени в Оксфордском университете в 1982 году и степень доктора философии по информатике в Станфордском университете в 1986 году. Затем он перешел на факультет Калифорнийского университета в г. Беркли, где занял должность профессора компьютерных наук, директора предприятия Center for Intelligent Systems и заведующего кафедрой инженерного искусства Смита–Задэ. В 1990 году он получил от фонда National Science Foundation премию Presidential Young Investigator Award, а в 1995 разделил первое место в конкурсе Computers and Thought Award. В 1996 году он победил на конкурсе Miller Research Professor в Калифорнийском университете, а в 2000 году был выдвинут на премию Chancellor's Professorship. В 1998 году он прочитал мемориальные лекции в память Форсита в Станфордском университете. Рассел — член AAAI (American Association for Artificial Intelligence — Американская ассоциация специалистов по искусственному интеллекту) и бывший член исполнительного совета этой ассоциации. Им опубликовано больше 100 статей по широкому кругу проблем искусственного интеллекта. Он также является автором книг *Use of Knowledge in Analogy and Induction* и *Do the Right Thing: Studies in Limited Rationality* (написана в соавторстве с Эриком Вефолдом — Eric Wefald).

Питер Норвиг — директор подразделения Search Quality компании Google, а также член AAAI и член исполнительного совета этой ассоциации. Перед этим он возглавлял подразделение Computational Sciences Division в исследовательском центре NASA Ames Research Center, а также руководил проводимыми в комитете NASA (National Aeronautics and Space Administration — Национальный комитет по аeronавтике и исследованию космического пространства) исследованиями и разработками по искусственному интеллекту и робототехнике. Еще раньше он занимал должность руководителя исследовательских работ в компании Junglee, где под его руководством была разработана одна из первых служб извлечения информации в Internet, и должность старшего научного сотрудника в подразделении компании Sun Microsystems Laboratories, которое работало над интеллектуальным информационным поиском. Он получил степень бакалавра наук по прикладной математике в Университете Брауна и степень доктора философии по информатике в Калифорнийском университете, г. Беркли. Он занимал должность профессора в Университете Южной Калифорнии и работал на факультете научно-технических исследований в Беркли. Ему принадлежат больше 50 публикаций по компьютерным наукам, в том числе книги *Case Studies in Common Lisp*, *Verbmobil: A Translation System for Face-to-Face Dialog* и *Intelligent Help Systems for UNIX*.

## **ЖДЕМ ВАШИХ ОТЗЫВОВ!**

---

Вы, уважаемый читатель, и есть главный критик и комментатор этой книги. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш Web-сервер и оставить свои замечания. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@williamspublishing.com

WWW: <http://www.williamspublishing.com>

Информация для писем:

из России: 115419, Москва, а/я 783

из Украины: 03150, Киев, а/я 152

## Часть I

# ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Введение	34
Интеллектуальные агенты	75

# 1 ВВЕДЕНИЕ

*В этой главе авторы пытаются объяснить, почему они рассматривают искусственный интеллект как тему, в наибольшей степени заслуживающую изучения, а также определить, в чем именно заключается данная тема; эти задачи необходимо решить, прежде чем приступать к дальнейшей работе.*

Люди называют себя *Homo sapiens* (человек разумный), поскольку для них мыслительные способности имеют очень важное значение. В течение тысяч лет человек пытается понять, как он думает, т.е. разобраться в том, как именно ему, сравнительно небольшому материальному объекту, удается ощущать, понимать, предсказывать и управлять миром, намного более значительным по своим размерам и гораздо более сложным по сравнению с ним. В области **искусственного интеллекта** (ИИ) решается еще более ответственная задача: специалисты в этой области пытаются не только понять природу интеллекта, но и создать интеллектуальные сущности.

Искусственный интеллект — это одна из новейших областей науки. Первые работы в этой области начались вскоре после Второй мировой войны, а само ее название было предложено в 1956 году. Ученые других специальностей чаще всего указывают искусственный интеллект, наряду с молекулярной биологией, как “область, в которой я больше всего хотел бы работать”. Студенты-физики вполне обоснованно считают, что все великие открытия в их области уже были сделаны Галилеем, Ньютоном, Эйнштейном и другими учеными. Искусственный интеллект, с другой стороны, все еще открывает возможности для проявления талантов нескольких настоящих Эйнштейнов.

В настоящее время тематика искусственного интеллекта охватывает огромный перечень научных направлений, начиная с таких задач общего характера, как обучение и восприятие, и заканчивая такими специальными задачами, как игра в шахматы, доказательство математических теорем, сочинение поэтических произведений и диагностика заболеваний. В искусственном интеллекте систематизируются и автоматизируются интеллектуальные задачи и поэтому эта область касается любой сферы интеллектуальной деятельности человека. В этом смысле искусственный интеллект является поистине универсальной научной областью.

## 1.1. ОБЩЕЕ ОПРЕДЕЛЕНИЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Из сказанного выше можно сделать вывод, что искусственный интеллект представляет собой чрезвычайно интересную научную область. Но определение этого научного направления в настоящей книге еще не было дано. В табл. 1.1 приведены определения искусственного интеллекта, взятые из восьми научных работ. Эти определения можно классифицировать по двум основным категориям. Грубо говоря, формулировки, приведенные в верхней части таблицы, касаются мыслительных процессов и способов рассуждения, а в нижней части таблицы находятся формулировки, касающиеся поведения. В определениях, приведенных слева, успех измеряется в терминах достоверного воспроизведения способностей человека, а формулировки, находящиеся справа, характеризуют конечные достижения в той области трактовки идеальной концепции интеллектуальности, которую авторы настоящей книги предпочитают называть **рациональностью**. Система является рациональной, если она “все действия выполняет правильно”, при условии, что система обладает знаниями о том, что является правильным.

**Таблица 1.1. Некоторые определения искусственного интеллекта, распределенные по четырем категориям**

Системы, которые думают подобно людям	Системы, которые думают рационально
“Новое захватывающее направление работ по созданию компьютеров, способных думать, ... машин, обладающих разумом, в полном и буквальном смысле этого слова” [631]	“Изучение умственных способностей с помощью вычислительных моделей” [239]
“[Автоматизация] действий, которые мы ассоциируем с человеческим мышлением, т.е. таких действий, как принятие решений, решение задач, обучение...” [95]	“Изучение таких вычислений, которые позволяют чувствовать, рассуждать и действовать” [1603]
Системы, которые действуют подобно людям	Системы, которые действуют рационально
“Искусство создания машин, которые выполняют функции, требующие интеллектуальности при их выполнении людьми” [871]	“Вычислительный интеллект — это наука о проектировании интеллектуальных агентов” [1227]
“Наука о том, как научить компьютеры делать то, в чем люди в настоящее время их превосходят” [1285]	“Искусственный интеллект... — это наука, посвященная изучению интеллектуального поведения артефактов!” [1146]

История развития искусственного интеллекта показывает, что интенсивные исследования проводились по всем четырем направлениям. Вполне можно предположить, что между теми учеными, которые в основном исходят из способностей людей, и теми, кто занимается главным образом решением проблемы рациональности, существуют определенные разногласия<sup>2</sup>. Подход, ориентированный на изучение человека, должен представлять собой эмпирическую научную область, развитие кото-

<sup>1</sup> Артефакт — искусственный объект.

<sup>2</sup> Необходимо указать, что авторы, проводя различие между человеческим и рациональным поведением, отнюдь не имеют в виду то, что люди обязательно действуют “нерационально” в том смысле этого слова, который характеризуется как “эмоциональная неустойчивость” или “неразумность”. Просто следует всегда помнить о том, что люди не идеальны, например, не все они становятся шахматными гроссмейстерами, даже если досконально изучили правила игры в шахматы, и, к сожалению, далеко не каждый получает высшие оценки на экзаменах. Некоторые систематические ошибки в человеческих рассуждениях были изучены и описаны в [762].

рой происходит по принципу выдвижения гипотез и их экспериментального подтверждения. С другой стороны, подход, основанный на понятии рациональности, представляет собой сочетание математики и техники. Каждые из этих групп ученых действуют разрозненно, но вместе с тем помогают друг другу. Ниже четыре указанных подхода рассматриваются более подробно.

### **Проверка того, способен ли компьютер действовать подобно человеку: подход, основанный на использовании теста Тьюринга**

❖ **Тест Тьюринга**, предложенный Аланом Тьюрингом [1520], был разработан в качестве удовлетворительного функционального определения интеллекта. Тьюринг решил, что нет смысла разрабатывать обширный список требований, необходимых для создания искусственного интеллекта, который к тому же может оказаться противоречивым, и предложил тест, основанный на том, что поведение объекта, обладающего искусственным интеллектом, в конечном итоге нельзя будет отличить от поведения таких бесспорно интеллектуальных существ, как человеческие существа. Компьютер успешно пройдет этот тест, если человек-экспериментатор, задавший ему в письменном виде определенные вопросы, не сможет определить, получены ли письменные ответы от другого человека или от некоторого устройства. В главе 26 подробно обсуждается этот тест и рассматривается вопрос о том, действительно ли можно считать интеллектуальным компьютером, который успешно прошел подобный тест. На данный момент просто отметим, что решение задачи по составлению программы для компьютера для того, чтобы он прошел этот тест, требует большого объема работы. За программируемый таким образом компьютер должен обладать перечисленными ниже возможностями.

- Средства ❖ **обработки текстов на естественных языках** (Natural Language Processing —NLP), позволяющие успешно общаться с компьютером, скажем на английском языке.
- Средства ❖ **представления знаний**, с помощью которых компьютер может записать в память то, что он узнает или прочитает.
- Средства ❖ **автоматического формирования логических выводов**, обеспечивающие возможность использовать хранимую информацию для поиска ответов на вопросы и вывода новых заключений.
- Средства ❖ **машинного обучения**, которые позволяют приспособливаться к новым обстоятельствам, а также обнаруживать и экстраполировать признаки стандартных ситуаций.

В teste Тьюринга сознательно исключено непосредственное физическое взаимодействие экспериментатора и компьютера, поскольку для создания искусственного интеллекта не требуется физическая имитация человека. Но в так называемом ❖ **полном teste Тьюринга** предусмотрено использование видеосигнала для того, чтобы экспериментатор мог проверить способности испытуемого объекта к восприятию, а также имел возможность представить физические объекты “в неполном виде” (пропустить их “через штриховку”). Чтобы пройти полный тест Тьюринга, компьютер должен обладать перечисленными ниже способностями.

- **❖ Машинное зрение** для восприятия объектов.
- Средства **❖ робототехники** для манипулирования объектами и перемещения в пространстве.

Шесть направлений исследований, перечисленных в данном разделе, составляют основную часть искусственного интеллекта, а Тьюринг заслуживает нашей благодарности за то, что предложил такой тест, который не потерял своей значимости и через 50 лет. Тем не менее исследователи искусственного интеллекта практически не занимаются решением задачи прохождения теста Тьюринга, считая, что гораздо важнее изучить основополагающие принципы интеллекта, чем продублировать одного из носителей естественного интеллекта. В частности, проблему “искусственного полета” удалось успешно решить лишь после того, как братья Райт и другие исследователи перестали имитировать птиц и приступили к изучению аэродинамики. В научных и технических работах по воздухоплаванию цель этой области знаний не определяется как “создание машин, которые в своем полете настолько напоминают голубей, что даже могут обмануть настоящих птиц”.

### **Как мыслить по-человечески: подход, основанный на когнитивном моделировании**

Прежде чем утверждать, что какая-то конкретная программа мыслит, как человек, требуется иметь некоторый способ определения того, как же мыслят люди. Необходимо проникнуть в сам фактически происходящий процесс работы человеческого разума. Для этого могут использоваться два способа: интроспекция (попытка проследить за ходом собственных мыслей) и психологические эксперименты. Только после создания достаточно точной теории мышления появится возможность представить формулы этой теории в виде компьютерной программы. И если входные и выходные данные программы, а также распределение выполняемых ею действий во времени будут точно соответствовать поведению человека, это может свидетельствовать о том, что некоторые механизмы данной программы могут также действовать в человеческом мозгу. Например, Аллен Ньюэлл (Allen Newell) и Герберт Саймон (Herbert Simon), которые разработали программу GPS (“General Problem Solver”—универсальный решатель задач) [1129], не стремились лишь к тому, чтобы эта программа правильно решала поставленные задачи. Их в большей степени забортило, чтобы запись этапов проводимых ею рассуждений совпадала с регистрацией рассуждений людей, решающих такие же задачи. В междисциплинарной области **❖ когнитологии** совместно используются компьютерные модели, взятые из искусственного интеллекта, и экспериментальные методы, взятые из психологии, для разработки точных и обоснованных теорий работы человеческого мозга.

Такая область знаний, как когнитология, является весьма увлекательной и настолько обширной, что ей вполне может быть посвящена отдельная энциклопедия [1599]. В данной книге авторы не пытаются описать все, что известно о человеческом познании. В ней лишь в некоторых местах комментируются аналогии или различия между методами искусственного интеллекта и человеческим познанием. Тем не менее настоящая научная когнитология обязательно должна быть основана на экспериментальном исследовании реальных людей или животных, а авторы данной книги предполагают, что ее читатель имеет доступ для экспериментирования только к компьютеру.

На начальных стадиях развития искусственного интеллекта часто возникала путаница между описанными выше подходами, например, иногда приходилось сталкиваться с такими утверждениями некоторых авторов, что предложенный ими алгоритм хорошо справляется с определенной задачей и поэтому является хорошей моделью способностей человека, или наоборот. Современные авторы излагают результаты своих исследований в этих двух областях отдельно; такое разделение позволяет развиваться быстрее как профессиональному интеллекту, так и когнитологии. Но эти две научные области продолжают обогащать друг друга, особенно в таких направлениях, как зрительное восприятие и понимание естественного языка. В последнее время особенно значительные успехи достигнуты в области зрительного восприятия благодаря использованию интегрированного подхода, в котором применяются и нейрофизиологические экспериментальные данные, и вычислительные модели.

### **Как мыслить рационально: подход, основанный на использовании “законов мышления”**

Греческий философ Аристотель был одним из первых, кто попытался определить законы “правильного мышления”, т.е. процессы формирования неопровергимых рассуждений. Его **силлогизмы** стали образцом для создания процедур доказательства, которые всегда позволяют прийти к правильным заключениям, если даны правильные предпосылки, например “Сократ — человек; все люди смертны; следовательно, Сократ смертен”. В основе этих исследований лежало предположение, что такие законы мышления управляют работой ума; на их основе развилось научное направление, получившее название **логика**.

В XIX столетии ученые, работавшие в области логики, создали точную систему логических обозначений для утверждений о предметах любого рода, которые встречаются в мире, и об отношениях между ними. (Сравните ее с обычной системой арифметических обозначений, которая предназначена в основном для формирования утверждений о равенстве и неравенстве чисел.) К 1965 году были уже разработаны программы, которые могли в принципе решить любую разрешимую проблему, описанную в системе логических обозначений<sup>3</sup>. Исследователи в области искусственного интеллекта, придерживающиеся так называемых традиций **логицизма**, надеются, что им удастся создать интеллектуальные системы на основе подобных программ.

Но при осуществлении указанного подхода возникают два серьезных препятствия. Во-первых, довольно сложно взять любые неформальные знания и выразить их в формальных терминах, требуемых для системы логических обозначений, особенно если эти знания не являются полностью достоверными. Во-вторых, возможность сравнительно легко решить проблему “в принципе” отнюдь не означает, что это действительно удастся сделать на практике. Даже такие задачи, в основе которых лежит несколько десятков фактов, могут исчерпать вычислительные ресурсы любого компьютера, если не используются определенные методы управления тем, какие этапы проведения рассуждений должны быть опробованы в первую очередь. Хотя с обоими этими препятствиями приходится сталкиваться при любой попытке создания вычислительных систем для автоматизации процесса проведения рассуждений, они были впервые обнаружены в рамках традиций логицизма.

---

<sup>3</sup> Если же решение не существует, программа может так и не остановиться в процессе его поиска.

## Как мыслить рационально: подход, основанный на использовании рационального агента

❖ **Агентом** считается все, что действует (слово *агент* произошло от из латинского слова *agere* — действовать). Но предполагается, что компьютерные агенты обладают некоторыми другими атрибутами, которые отличают их от обычных “программ”, такими как способность функционировать под автономным управлением, воспринимать свою среду, существовать в течение продолжительного периода времени, адаптироваться к изменениям и обладать способностью взять на себя достижение целей, поставленных другими. ❖ **Рациональным агентом** называется агент, который действует таким образом, чтобы можно было достичь наилучшего результата или, в условиях неопределенности, наилучшего ожидаемого результата.

В подходе к созданию искусственного интеллекта на основе “законов мышления” акцент был сделан на формировании правильных логических выводов. Безусловно, иногда формирование правильных логических выводов становится и частью функционирования рационального агента, поскольку один из способов рациональной организации своих действий состоит в том, чтобы логическим путем прийти к заключению, что данное конкретное действие позволяет достичь указанных целей, а затем действовать в соответствии с принятым решением. С другой стороны, правильный логический вывод не исчерпывает понятия рациональности, поскольку часто возникают ситуации, в которых невозможно однозначно выбрать какие-либо правильные действия, но все равно надо что-то делать. Кроме того, существуют способы рациональной организации действий, в отношении которых нельзя утверждать, что в них используется логический вывод. Например, отдергивание пальца от горячей печи — это рефлекторное действие, которое обычно является более успешным по сравнению с более медленным движением, сделанным после тщательного обдумывания всех обстоятельств.

Таким образом, все навыки, требуемые для прохождения теста Тьюринга, позволяют также осуществлять рациональные действия. Итак, прежде всего необходимо иметь возможность представлять знания и проводить на основании них рассуждения, поскольку это позволяет вырабатывать приемлемые решения в самых различных ситуациях. Необходимо обладать способностью формировать понятные предложения на естественном языке, поскольку в сложный социум принимают только тех, кто способен правильно высказывать свои мысли. Необходимо учиться не только ради приобретения эрудиции, но и в связи с тем, что лучшее представление о том, как устроен мир, позволяет вырабатывать более эффективные стратегии действий в этом мире. Нужно обладать способностью к зрительному восприятию не только потому, что процесс визуального наблюдения позволяет получать удовольствие, но и потому, что зрение подсказывает, чего можно достичь с помощью определенного действия, например тот, кто сумеет быстрее всех разглядеть лакомый кусочек, получит шанс подобраться к нему раньше других.

По этим причинам подход к исследованию искусственного интеллекта как области проектирования рациональных агентов имеет, по меньшей мере, два преимущества. Во-первых, этот подход является более общим по сравнению с подходом, основанном на использовании “законов мышления”, поскольку правильный логический вывод — это просто один из нескольких возможных механизмов достижения рациональности. Во-вторых, он является более перспективным для научной разработки по

сравнению с подходами, основанными на изучении человеческого поведения или человеческого мышления, поскольку стандарт рациональности четко определен и полностью обобщен. Человеческое поведение, с другой стороны, хорошо приспособлено лишь для одной определенной среды и отчасти является продуктом сложного и в основном неизученного эволюционного процесса, который, как оказалось, отнюдь не позволяет формировать существа, идеальные во всех отношениях.

*Поэтому данная книга в основном посвящена описанию общих принципов работы рациональных агентов и компонентов, необходимых для их создания.* Из изложенного в ней станет очевидно, что несмотря на кажущуюся простоту формулировки этой проблемы, при попытке ее решения возникает невероятное количество трудностей. Некоторые из этих трудностей более подробно описываются в главе 2.

Следует всегда учитывать одно важное замечание: нужно неизменно исходить из того, что в сложной среде задача достижения идеальной рациональности, при которой всегда выполняются правильные действия, не осуществима. Дело в том, что при этом предъявляются слишком высокие требования к вычислительным ресурсам. Но в основной части данной книги применяется рабочая гипотеза, согласно которой идеальная рациональность является хорошей отправной точкой для анализа. Такой подход позволяет упростить задачу создания рационального агента и предоставляет подходящую основу для описания большей части теоретического материала в этой области. В главах 6 и 17 речь идет непосредственно о проблеме **ограниченной рациональности** — организации приемлемых действий в тех ситуациях, когда не хватает времени на выполнение всех вычислений, которые действительно могли бы потребоваться.

## **1.2. ПРЕДЫСТОРИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА**

---

В данном разделе кратко описана история развития научных дисциплин, которые внесли свой вклад в область искусственного интеллекта в виде конкретных идей, взглядов и методов. Как и в любом историческом очерке, поневоле приходится ограничиваться описанием небольшого круга людей, событий и открытий, игнорируя все остальные факты, которые были не менее важными. Авторы построили этот исторический экскурс вокруг ограниченного круга вопросов. Безусловно, они не хотели бы, чтобы у читателя создалось такое впечатление, будто эти вопросы являются единственными, которые рассматриваются в указанных научных дисциплинах, или что сами эти дисциплины развивались исключительно ради того, чтобы их конечным итогом стало создание искусственного интеллекта.

### **Философия (период с 428 года до н.э. по настоящее время)**

- Могут ли использоваться формальные правила для вывода правильных заключений?
- Как такой идеальный объект, как мысль, рождается в таком физическом объекте, как мозг?
- Каково происхождение знаний?
- Каким образом знания ведут к действиям?

Точный свод законов, руководящих рациональной частью мышления, был впервые сформулирован Аристотелем (384–322 годы до н.э.). Он разработал неформализованную систему силлогизмов, предназначенную для проведения правильных рассуждений, которая позволяла любому вырабатывать логические заключения механически, при наличии начальных предпосылок. Гораздо позднее Раймунд Луллий (умер в 1315 году) выдвинул идею, что полезные рассуждения можно фактически проводить с помощью механического артефакта. Предложенные им “концептуальные колеса” показаны на обложке данной книги. Томас Гоббс (1588–1679) предположил, что рассуждения аналогичны числовым расчетам и что “в наших неслышимых мыслях мы по неволе складываем и вычитаем”. В то время автоматизация самих вычислений уже шла полным ходом; примерно в 1500 году Леонардо да Винчи (1452–1519) спроектировал, но не построил механический калькулятор; недавно проведенная реконструкция показала, что его проект является работоспособным. Первая известная вычислительная машина была создана примерно в 1623 году немецким ученым Вильгельмом Шиккардом (1592–1635), хотя более известна машина Паскалина, построенная в 1642 году Блезом Паскалем (1623–1662). Паскаль писал, что “арифметическая машина производит эффект, который кажется более близким к мышлению по сравнению с любыми действиями животных”. Готтфрид Вильгельм Лейбниц (1646–1716) создал механическое устройство, предназначенное для выполнения операций над понятиями, а не над числами, но область его действия была довольно ограниченной.

После того как человечество осознало, каким должен быть набор правил, способных описать формальную, рациональную часть мышления, следующим этапом оказалось то, что разум стал рассматриваться как физическая система. Рене Декарт (1596–1650) впервые опубликовал результаты обсуждения различий между разумом и материей, а также возникающих при этом проблем. Одна из проблем, связанных с чисто физическими представлениями о разуме, состоит в том, что они, по-видимому, почти не оставляют места для свободной воли: ведь если разум руководствуется исключительно физическими законами, то человек проявляет не больше свободной воли по сравнению с булыжником, “решившим” упасть в направлении к центру земли. Несмотря на то что Декарт был убежденным сторонником взглядов, признающих только власть разума, он был также приверженцем  **дуализма**. Декарт считал, что существует такая часть человеческого разума (душа, или дух), которая находится за пределами естества и не подчиняется физическим законам. С другой стороны, животные не обладают таким дуалистическим свойством, поэтому их можно рассматривать как своего рода машины. Альтернативой дуализму является  **материализм**, согласно которому разумное поведение складывается из операций, выполняемых мозгом в соответствии с законами физики. *Свободная воля* — это просто форма, в которую в процессе выбора преобразуется восприятие доступных вариантов.

Если предположить, что знаниями манипулирует физический разум, то возникает следующая проблема — установить источник знаний. Такое научное направление, как  **эмпиризм**, родоначальником которого был Фрэнсис Бэкон (1561–1626), автор *Нового Органона*<sup>4</sup>, можно охарактеризовать высказыванием Джона Локка (1632–1704): “В человеческом понимании нет ничего, что не проявлялось бы прежде всего в ощущениях”. Дэвид Юм (1711–1776) в своей книге *A Treatise of Human Nature*

<sup>4</sup> Эта книга была выпущена как новая версия *Органона* (или инструмента мышления) Аристотеля.

(Трактат о человеческой природе) [705] предложил метод, известный теперь под названием **принципа индукции**, который состоит в том, что общие правила вырабатываются путем изучения повторяющихся ассоциаций между элементами, которые рассматриваются в этих правилах. Основываясь на работе Людвига Витгенштейна (1889–1951) и Бертрана Рассела (1872–1970), знаменитый Венский кружок, возглавляемый Рудольфом Карнапом (1891–1970), разработал доктрину **логического позитивизма**. Согласно этой доктрине все знания могут быть охарактеризованы с помощью логических теорий, связанных в конечном итоге с **констатирующими предложениями**, которые соответствуют входным сенсорным данным<sup>5</sup>. В **теории подтверждения** Рудольфа Карнапа и Карла Хемпеля (1905–1997) предпринята попытка понять, как знания могут быть приобретены из опыта. В книге Карнапа *The Logical Structure of the World* [223] определена явно заданная вычислительная процедура для извлечения знаний из результатов элементарных опытов. По-видимому, это — первая теория мышления как вычислительного процесса.

Заключительным элементом в этой картине философских исследований проблемы разума является связь между знаниями и действиями. Данный вопрос для искусственного интеллекта является жизненно важным, поскольку интеллектуальность требует не только размышлений, но и действий. Кроме того, только поняв способы обоснования действий, можно понять, как создать агента, действия которого будут обоснованными (или рациональными). Аристотель утверждал, что действия обоснованы логической связью между целями и знаниями о результатах данного конкретного действия (последняя часть приведенной ниже цитаты Аристотеля на языке оригинала размещена также на обложке данной книги). Характерным примером рассуждений о рациональных действиях являются следующие.

Но почему происходит так, что размышления иногда сопровождаются действием, а иногда — нет, иногда за ними следует движение, а иногда — нет? Создается впечатление, как будто почти то же самое происходит и в случае построения рассуждений и формирования выводов о неизменных объектах. Но в таком случае целью умственной деятельности оказывается умозрительное суждение..., тогда как заключением, которое следует из данных двух предпосылок, является действие... Мне нужна защита от дождя; защитой может послужить плащ. Мне нужен плащ. Я должен сам изготовить то, в чем я нуждаюсь; я нуждаюсь в плаще. Я должен изготовить плащ. И заключение “я должен изготовить плащ” становится действием ([1151, с. 40]).

В книге *Никомахова этика* (том III. 3, 1112б) Аристотеля можно найти более подробные рассуждения на эту тему, где также предложен алгоритм.

Нам предоставляется право выбора не целей, а средств достижения цели, ведь врач рассуждает не о том, должен ли он лечить, а оратор — не о том, станет ли он убеждать... Поставив цель, он размышляет, как и какими средствами ее достичь; а если окажется несколько средств, то определяет, какое из них самое простое и наилучшее; если же достижению цели служит одно средство, думает, как ее достичь при помощи этого средства и что будет средством для этого средства, пока не дойдет до первой причины, которую находит последней... и то, что было последним в порядке анализа, обычно становится первым в порядке осуществления... Если же он приходит к выводу, что цель недостижима, отступает-

<sup>5</sup> В данной картине мира все осмысленные утверждения можно подтвердить или опровергнуть либо с помощью анализа смысла слов, либо путем проведения экспериментов. Поскольку при этом основная часть метафизики остается за бортом, в чем и состояло намерение создателей данного направления, логический позитивизм в некоторых кругах встретил неодобрительное отношение.

ся, например, если нужны деньги, а достать их нельзя; но если достижение цели кажется возможным, то пытается ее достичь.

Алгоритм Аристотеля был реализован через 2300 лет Ньюэллом и Саймоном в программе GPS. Теперь то, что создано на его базе, принято называть *регрессивной системой планирования* (см. главу 11).

Анализ на основе цели является полезным, но не дает ответа на то, что делать, если к цели ведет несколько вариантов действий или ни один вариант действий не позволяет достичь ее полностью. Антуан Арно (1612–1694) правильно описал количественную формулу для принятия решения о том, какое действие следует предпринять в подобных случаях (см. главу 16). В книге *Utilitarianism* приверженца утилитаризма Джона Стюарта Милла (1806–1873) [1050] провозглашена идея о том, что критерии принятия рациональных решений должны применяться во всех сферах человеческой деятельности. Более формальная теория принятия решений рассматривается в следующем разделе.

### Математика (период примерно с 800 года по настоящее время)

- Каковы формальные правила формирования правильных заключений?
- Как определить пределы вычислимости?
- Как проводить рассуждения с использованием недостоверной информации?

Философы сформулировали наиболее важные идеи искусственного интеллекта, но для преобразования его в формальную науку потребовалось достичь определенного уровня математической формализации в трех фундаментальных областях: логика, вычисления и вероятность.

Истоки идей формальной логики можно найти в работах философов древней Греции (см. главу 7), но ее становление как математической дисциплины фактически началась с трудов Джорджа Буля (1815–1864), который детально разработал логику высказываний, или булеву логику [149]. В 1879 году Готтлоб Фреге (1848–1925) расширил булеву логику для включения в нее объектов и отношений, создав логику первого порядка, которая в настоящее время используется как наиболее фундаментальная система представления знаний<sup>6</sup>. Альфред Тарский (1902–1983) впервые ввел в научный обиход теорию ссылок, которая показывает, как связать логические объекты с объектами реального мира. Следующий этап состоял в определении пределов того, что может быть сделано с помощью логики и вычислений.

Первым нетривиальным  алгоритмом считается алгоритм вычисления наибольшего общего знаменателя, предложенный Евклидом. Исследование алгоритмов как самостоятельных объектов было начато аль-Хорезми, среднеазиатским математиком IX столетия, благодаря работам которого Европа познакомилась с арабскими цифрами и алгеброй. Буль и другие ученые широко обсуждали алгоритмы логического вывода, а к концу XIX столетия уже предпринимались усилия по формализации общих принципов проведения математических рассуждений как логического вывода. В 1900 году Давид Гильберт (1862–1943) представил список из 23 проблем и правильно предсказал, что эти проблемы будут занимать математиков почти до кон-

<sup>6</sup> Предложенная Готтлобом Фреге система обозначений для логики первого порядка так и не нашла широкого распространения по причинам, которые становятся сразу же очевидными из примера, приведенного на первой странице обложки.

ца XX века. Последняя из этих проблем представляет собой вопрос о том, существует ли алгоритм для определения истинности любого логического высказывания, в состав которого входят натуральные числа. Это — так называемая знаменитая проблема поиска решения (*Entscheidungsproblem*). По сути, этот вопрос, заданный Гильбертом, сводился к определению того, есть ли фундаментальные пределы, ограничивающие мощь эффективных процедур доказательства. В 1930 году Курт Гёдель (1906–1978) показал, что существует эффективная процедура доказательства любого истинного высказывания в логике первого порядка Фрэгера и Рассела, но при этом логика первого порядка не позволяет выразить принцип математической индукции, необходимый для представления натуральных чисел. В 1931 году Гёдель показал, что действительно существуют реальные пределы вычислимости. Предложенная им **теорема о неполноте** показывает, что в любом языке, достаточно выразительном для описания свойств натуральных чисел, существуют истинные высказывания, которые являются недоказуемыми, в том смысле, что их истинность невозможно установить с помощью какого-либо алгоритма.

Этот фундаментальный результат может также рассматриваться как демонстрация того, что имеются некоторые функции от целых чисел, которые не могут быть представлены с помощью какого-либо алгоритма, т.е. они не могут быть вычислены. Это побудило Алана Тьюринга (1912–1954) попытаться точно охарактеризовать, какие функции способны быть вычисленными. Этот подход фактически немного проблематичен, поскольку в действительности понятию вычисления, или эффективной процедуры вычисления, не может быть дано формальное определение. Но общепризнано, что вполне удовлетворительное определение дано в тезисе Чёрча–Тьюринга, который указывает, что машина Тьюринга [1518] способна вычислить любую вычислимую функцию. Кроме того, Тьюринг показал, что существуют некоторые функции, которые не могут быть вычислены машиной Тьюринга. Например, вообще говоря, ни одна машина не способна определить, возвратит ли данная конкретная программа ответ на конкретные входные данные или будет работать до бесконечности.

Хотя для понимания возможностей вычисления очень важны понятия недоказуемости и невычислимости, гораздо большее влияние на развитие искусственного интеллекта оказало понятие **неразрешимости**. Грубо говоря, задача называется *неразрешимой*, если время, требуемое для решения отдельных экземпляров этой задачи, растет экспоненциально с увеличением размеров этих экземпляров. Различие между полиномиальным и экспоненциальным ростом сложности было впервые подчеркнуто в середине 1960-х годов в работах Кобхэма [272] и Эдмондса [430]. Важность этого открытия состоит в следующем: экспоненциальный рост означает, что даже экземпляры задачи умеренной величины не могут быть решены за какое-либо приемлемое время. Поэтому, например, приходится заниматься разделением общей задачи выработки интеллектуального поведения на разрешимые подзадачи, а не пытаться решать неразрешимую задачу.

Как можно распознать неразрешимую проблему? Один из приемлемых методов такого распознавания представлен в виде теории **NP-полноты**, впервые предложенной Стивеном Куком [289] и Ричардом Карпом [772]. Кук и Карп показали, что существуют большие классы канонических задач комбинаторного поиска и формирования рассуждений, которые являются NP-полными. Существует вероятность того, что любой класс задач, к которому сводится этот класс NP-полных задач, является неразрешимым. (Хотя еще не было доказано, что NP-полные задачи обязатель-

но являются неразрешимыми, большинство теоретиков считают, что дело обстоит именно так.) Эти результаты контрастируют с тем оптимизмом, с которым в популярных периодических изданиях приветствовалось появление первых компьютеров под такими заголовками, как “Электронные супермозги”, которые думают “быстрее Эйнштейна!” Несмотря на постоянное повышение быстродействия компьютеров, характерной особенностью интеллектуальных систем является экономное использование ресурсов. Грубо говоря, наш мир, в котором должны освоиться системы ИИ, — это чрезвычайно крупный экземпляр задачи. В последние годы методы искусственного интеллекта помогли разобраться в том, почему некоторые экземпляры NP-полных задач являются сложными, а другие простыми [244].

Кроме логики и теории вычислений, третий по величине вклад математиков в искусственный интеллект состоял в разработке теории вероятностей. Идея вероятности была впервые сформулирована итальянским математиком Джероламо Кардано (1501–1576), который описал ее в терминах результатов событий с несколькими исходами, возникающих в азартных играх. Теория вероятностей быстро стала неотъемлемой частью всех количественных наук, помогая использовать недостоверные результаты измерений и неполные теории. Пьер Ферма (1601–1665), Блез Паскаль (1623–1662), Джеймс Бернуlli (1654–1705), Пьер Лаплас (1749–1827) и другие ученые внесли большой вклад в эту теорию и ввели новые статистические методы. Томас Байес (1702–1761) предложил правило обновления вероятностей с учетом новых фактов. Правило Байеса и возникшее на его основе научное направление, называемое *байесовским анализом*, лежат в основе большинства современных подходов к проведению рассуждений с учетом неопределенности в системах искусственного интеллекта.

### Экономика (период с 1776 года по настоящее время)

- Как следует организовать принятие решений для максимизации вознаграждения?
- Как действовать в таких условиях, когда другие могут препятствовать осуществлению намеченных действий?
- Как действовать в таких условиях, когда вознаграждение может быть предоставлено лишь в отдаленном будущем?

Экономика как наука возникла в 1776 году, когда шотландский философ Адам Смит (1723–1790) опубликовал свою книгу *An Inquiry into the Nature and Causes of the Wealth of Nations* (Исследование о природе и причинах богатства народов). Важный вклад в экономику был сделан еще древнегреческими учеными и другими предшественниками Смита, но только Смит впервые сумел оформить эту область знаний как науку, используя идею, что любую экономику можно рассматривать как состоящую из отдельных агентов, стремящихся максимизировать свое собственное экономическое благосостояние. Большинство людей считают, что экономика посвящена изучению денежного оборота, но любой экономист ответит на это, что в действительности он изучает то, как люди делают выбор, который ведет к предпочтительным для них результатам. Математическая трактовка понятия “предпочтительных результатов”, или **полезности**, была впервые formalизована Леоном Валрасом (1834–1910), уточнена Фрэнком Рамсеем [1265], а затем усовершенствована Джоном фон Нейманом и Оскаром Моргенштерном в книге *The Theory of Games and Economic Behavior* (Теория игр и экономического поведения) [1546].

❖ **Теория решений**, которая объединяет в себе теорию вероятностей и теорию полезности, предоставляет формальную и полную инфраструктуру для принятия решений (в области экономики или в другой области) в условиях неопределенности, т.е. в тех случаях, когда среда, в которой действует лицо, принимающее решение, наиболее адекватно может быть представлена лишь с помощью вероятностных описаний. Она хорошо подходит для “крупных” экономических образований, где каждый агент не обязан учитывать действия других агентов как индивидуумов. А в “небольших” экономических образованиях ситуация в большей степени напоминает **игру**, поскольку действия одного игрока могут существенно повлиять на полезность действий другого (или положительно, или отрицательно). ❖ **Теория игр**, разработанная фон Нейманом и Моргенштерном (см. также [963]), позволяет сделать неожиданный вывод, что в некоторых играх рациональный агент должен действовать случайным образом или, по крайней мере, таким образом, который кажется случайнym для соперников.

Экономисты чаще всего не стремятся найти ответ на третий вопрос, приведенный выше, т.е. не пытаются выработать способ принятия рациональных решений в тех условиях, когда вознаграждение в ответ на определенные действия не предоставляется немедленно, а становится результатом нескольких действий, выполненных в определенной последовательности. Изучению этой темы посвящена область ❖ **исследования операций**, которая возникла во время Второй мировой войны в результате усилий, которые были предприняты в Британии по оптимизации работы радарных установок, а в дальнейшем нашла применение и в гражданском обществе при выработке сложных управлеченческих решений. В работе Ричарда Беллмана [97] formalизован определенный класс последовательных задач выработки решений, называемых **марковскими процессами принятия решений** (Markov Decision Process — MDP), которые рассматриваются в главах 17 и 21.

Работы в области экономики и исследования операций оказали большое влияние на сформулированное в этой книге понятие рациональных агентов, но в течение многих лет исследования в области искусственного интеллекта проводились совсем по другим направлениям. Одной из причин этого была кажущаяся **сложность** задачи выработки рациональных решений. Тем не менее Герберт Саймон (1916–2001) в некоторых из своих ранних работ показал, что лучшее описание фактического поведения человека дают модели, основанные на ❖ **удовлетворении** (принятии решений, которые являются “достаточно приемлемыми”), а не модели, предусматривающие трудоемкий расчет оптимального решения [1414], и стал одним из первых исследователей в области искусственного интеллекта, получившим Нобелевскую премию по экономике (это произошло в 1978 году). В 1990-х годах наблюдалось возрождение интереса к использованию методов теории решений для систем агентов [1576].

### **Неврология (период с 1861 года по настоящее время)**

- Как происходит обработка информации в мозгу?

❖ **Неврология** — это наука, посвященная изучению нервной системы, в частности мозга. Одной из величайших загадок, не поддающихся научному описанию, остается определение того, как именно мозг обеспечивает мышление. Понимание того, что мышление каким-то образом связано с мозгом, существовало в течение тысяч лет, поскольку люди обнаружили, что сильные удары по голове могут привести к ум-

ственному расстройству. Кроме того, уже давно было известно, что человеческий мозг обладает какими-то важными особенностями; еще примерно в 335 до н.э. Аристотель<sup>7</sup> писал: “Из всех животных только человек имеет самый крупный мозг по сравнению с его размерами”. Тем не менее широкое признание того, что мозг является вместилищем сознания, произошло только в середине XVIII столетия. До этого в качестве возможных источников сознания рассматривались сердце, селезенка и шишковидная железа (эпифиз).

Исследования афазии (нарушения речи) у пациентов с повреждением мозга, проведенные Полем Броком (1824–1880) в 1861 году, снова пробудили интерес к этой научной области и послужили для многих представителей медицины доказательством существования в мозгу локализованных участков, ответственных за конкретные познавательные функции. Например, этот ученый показал, что функции формирования речи сосредоточены в той части левого полушария, которая теперь называется зоной Броока<sup>8</sup>. К тому времени уже было известно, что мозг состоит из нервных клеток, или ~~нейронов~~, но только в 1873 году Камилло Гольджи (1843–1926) сумел разработать надежный метод, позволяющий наблюдать за отдельными нейронами в мозгу (рис. 1.1). Этот метод использовал Сантьяго Рамон и Каахал (1852–1934) в своих пионерских исследованиях нейронных структур мозга<sup>9</sup>.

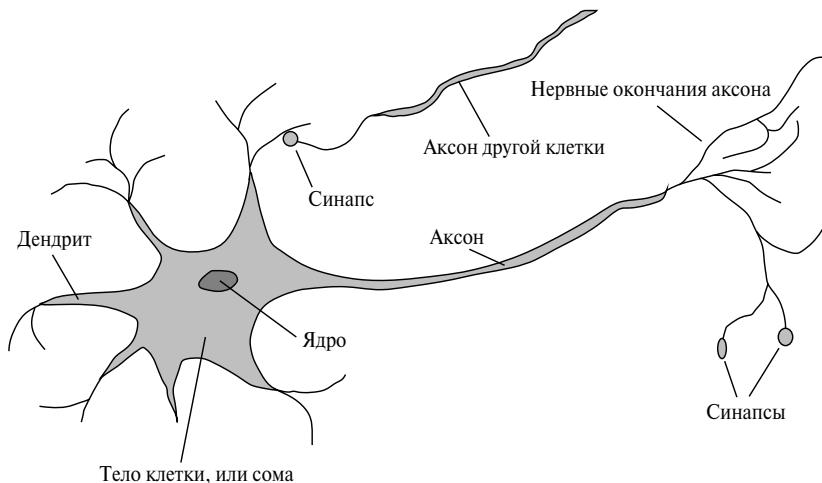
Теперь ученые располагают некоторыми данными о том, как связаны между собой отдельные области мозга и те части тела, которыми они управляют или от которых получают сенсорные данные. Оказалось, что подобная привязка может коренным образом измениться в течение нескольких недель, а у некоторых животных, по-видимому, имеется несколько вариантов такой привязки. Более того, еще не совсем понятно, как другие области могут взять на себя функции поврежденных областей. К тому же почти полностью отсутствуют обоснованные теории того, как осуществляется хранение информации в памяти индивидуума.

Измерение активности неповрежденного мозга началось в 1929 году с изобретения электроэнцефалографа (ЭЭГ) Гансом Бергером. Разработки в области получения изображений на основе функционального магнитного резонанса [1152] позволили неврологам получать исключительно подробные изображения активности мозга, что дает возможность проводить измерения характеристик физиологических процессов, которые связаны с происходящими познавательными процессами какими-то интересными способами. Эти возможности для исследований становятся еще более широкими благодаря прогрессу в области регистрации нейронной активности отдельной клетки. Но, несмотря на эти успехи, ученые еще очень далеки от понимания того, как действительно осуществляется любой из этих познавательных процессов.

<sup>7</sup> С тех пор было обнаружено, что некоторые виды дельфинов и китов имеют относительно более крупный мозг. Современные ученые считают, что большие размеры человеческого мозга отчасти обусловлены усовершенствованием системы его охлаждения на последних этапах эволюции человека.

<sup>8</sup> Многие цитируют в качестве возможного более раннего источника работу Александра Гуда [673].

<sup>9</sup> Гольджи упорно отстаивал свое мнение, что функции мозга осуществляются в основном непрерывной средой, в которую включены нейроны, тогда как Каахал проповедовал “нейронную доктрину”. Эти ученые совместно получили Нобелевскую премию в 1906 году, но в роли лауреатов произнесли речи, содержащие довольно антагонистичные взаимные выпады.



*Рис. 1.1. Части нервной клетки, или нейрона. Каждый нейрон состоит из тела клетки (или сомы), которое содержит ядро клетки. От тела клетки отвертываются множество коротких волокон, называемых дендритами, и одно длинное волокно, называемое аксоном. Аксон растягивается на большое расстояние, намного превышающее то, что показано в масштабе этого рисунка. Обычно аксоны имеют длину 1 см (что превышает в 100 раз диаметр тела клетки), но могут достигать 1 метра. Нейрон создает соединения с другими нейронами, количество которых может составлять от 10 до 100 000 в точках сопряжения, называемых синапсами. Сигналы распространяются от одного нейрона к другому с помощью сложной электрохимической реакции. Эти сигналы управляют активностью мозга в течение короткого интервала, а также становятся причиной долговременных изменений состояния самих нейронов и их соединений. Считается, что эти механизмы служат в мозгу основой для обучения. Обработка информации главным образом происходит в коре головного мозга, которая представляет собой самый внешний слой нейронов мозга. По-видимому, основной структурной единицей является столбец ткани, имеющий диаметр около 0,5 мм и протяженность на всю глубину коры, толщина которой в человеческом мозгу составляет около 4 мм. Каждый столбец содержит примерно 20 000 нейронов*

Тем не менее работы в области неврологии позволяют сделать поистине удивительное заключение о том, что ~~совместная работа простых клеток может приводить к появлению мышления, действия и сознания или, другими словами, что мозг порождает разум~~ [1379]. После этого открытия единственной реально существующей альтернативной теорией остается *мистицизм*, приверженцы которого провозглашают, что существует некое мистическое пространство, находящееся за пределами физического опыта, в котором функционирует разум.

Мозг и цифровой компьютер выполняют совершенно разные задачи и имеют различные свойства. В табл. 1.2 показано, что в типичном мозгу человека имеется в 1000 раз больше нейронов, чем логических элементов в процессоре типичного компьютера высокого класса. В соответствии с законом Мура<sup>10</sup> может быть сделан про-

<sup>10</sup> Закон Мура указывает, что плотность транзисторов в расчете на единицу площади удваивается примерно через каждые 1–1,5 года. Количество нейронов в мозгу человека, по расчетам, должно удваиваться примерно через каждые 2–4 миллиона лет.

гноз, что количество логических элементов в процессоре станет равным количеству нейронов в мозгу примерно к 2020 году. Безусловно, эти прогнозы мало о чем говорят; кроме того, это различие в отношении количества элементов является незначительным по сравнению с различием в скорости переключения и степени распараллеливания. Микросхемы компьютера способны выполнить отдельную команду меньше чем за наносекунду, тогда как нейроны действуют в миллионы раз медленнее. Но мозг сторицей восполняет этот свой недостаток, поскольку все его нейроны и синапсы действуют одновременно, тогда как большинство современных компьютеров имеет только один процессор или небольшое количество процессоров. Таким образом, ~~даже несмотря на то, что компьютер обладает преимуществом более чем в миллион раз в физической скорости переключения, оказывается, что мозг по сравнению с ним выполняет все свои действия примерно в 100 000 раз быстрее.~~

**Таблица 1.2. Грубое сравнение физических вычислительных ресурсов, имеющихся в компьютере (приблизительные оценки по состоянию на 2003 год) и в мозгу. Со времени выпуска первого издания данной книги показатели компьютера выросли, по меньшей мере, в 10 раз и будут, как ожидается, продолжать расти в течение текущего десятилетия. А показатели мозга за последние 10 000 лет не изменились**

	Компьютер	Человеческий мозг
Вычислительные модули	Один центральный процессор, $10^8$ логических элементов	$10^{11}$ нейронов
Модули памяти	Оперативная память на $10^{10}$ битов Диск емкостью $10^{11}$ битов	$10^{11}$ нейронов $10^{14}$ синапсов
Продолжительность цикла обработки	$10^{-9}$ секунды	$10^{-3}$ секунды
Пропускная способность	$10^{10}$ бит/с	$10^{14}$ бит/с
Количество обновлений памяти в секунду	$10^9$	$10^{14}$

## Психология (период с 1879 года по настоящее время)

- Как думают и действуют люди и животные?

Истоки научной психологии обычно прослеживаются до работ немецкого физика Германа фон Гельмгольца (1821–1894) и его студента Вильгельма Вундта (1832–1920). Гельмгольц применил научный метод для изучения зрения человека, и выпущенная им книга *Handbook of Physiological Optics* даже в наши дни характеризуется как “непревзойденный по своей важности вклад в изучение физики и физиологии зрения человека” [1111, с. 15]. В 1879 году Вундт открыл первую лабораторию по экспериментальной психологии в Лейпцигском университете. Вундт настаивал на проведении тщательно контролируемых экспериментов, в которых его сотрудники выполняли задачи по восприятию или формированию ассоциаций, проводя интроспективные наблюдения за своими мыслительными процессами. Такой тщательный контроль позволил ему сделать очень многое для превращения психологии в науку, но из-за субъективного характера данных вероятность того, что экспериментатор будет стремиться опровергнуть выдвинутые им теории, оставалась очень низкой. С другой стороны, биологи, изучающие поведение животных, не пользовались интроспективными данными и разработали объективную методологию, как показал Г.С. Дженнингс [733] в своей важной работе *Behavior of the Lower Organisms*. Распространяя этот подход на людей, сторонники ~~бихевиористского~~ движения, возглавляемые Джоном Уотсоном

(1878–1958), отвергали любую теорию, учитывающую мыслительные процессы, на том основании, что интроспекция не может предоставлять надежные свидетельства. Бихевиористы настаивали на том, что следует изучать только объективные меры восприятия (или стимулы), предъявленные животному, и вытекающие из этого действия (или отклики на стимулы). Такие мыслительные конструкции, как знания, убеждения, цели и последовательные рассуждения, отвергались как ненаучная “обывательская психология”. Бихевиоризм позволил многое узнать о крысах и голубях, но оказался менее успешным при изучении людей. Тем не менее это научное направление сохраняло за собой мощные позиции в области психологии (особенно в Соединенных Штатах Америки) в период примерно с 1920 по 1960 годы.

Взгляды, согласно которым мозг рассматривается как устройство обработки информации, характерные для представителей ~~когнитивной психологии~~, прослеживаются, по крайней мере, до работ Уильяма Джеймса<sup>11</sup> (1842–1910). Гельмгольц также утверждал, что восприятие связано с определенной формой подсознательного логического вывода. В Соединенных Штатах такой подход к изучению познавательных процессов был в основном отвергнут из-за широкого распространения бихевиористских взглядов, но на факультете прикладной психологии Кембриджского университета, возглавляемом Фредериком Бартлеттом (1886–1969), удалось организовать проведение широкого спектра работ в области когнитивного моделирования. В своей книге *The Nature of Explanation* студент и последователь Бартлетта, Кеннет Крэг [306], привел весомые доводы в пользу допустимости применения таких “мыслительных” терминов, как убеждения и цели, доказав, что они являются не менее научными, чем, скажем, такие термины, применяемые в рассуждениях о газах, как давление и температура, несмотря на то, что речь в них идет о молекулах, которые сами не обладают этими характеристиками. Крэг обозначил следующие три этапа деятельности агента, основанного на знаниях: во-первых, действующий стимул должен быть преобразован во внутреннее представление, во-вторых, с этим представлением должны быть выполнены манипуляции с помощью познавательных процессов для выработки новых внутренних представлений, и, в-третьих, они должны быть, в свою очередь, снова преобразованы в действия. Он наглядно объяснил, почему такой проект является приемлемым для любого агента.

Если живой организм несет в своей голове “модель в уменьшенном масштабе” внешней реальности и своих возможных действий, то обладает способностью проверять различные варианты, приходить к заключению, какой из них является наилучшим, реагировать на будущие ситуации, прежде чем они возникнут, использовать знания о прошлых событиях, сталкиваясь с настоящим и будущим, и во всех отношениях реагировать на опасности, встречаясь с ними, гораздо полнее, безопаснее для себя, а также в более компетентной форме [306].

В 1945 году, после смерти Крэга в результате несчастного случая во время катания на велосипеде, его работа была продолжена Дональдом Броудбентом, книга *Perception and Communication* [188] которого включила некоторые из первых моделей информационной обработки психологических феноменов. Между тем в Соединенных Штатах работы в области компьютерного моделирования привели к созданию такого

<sup>11</sup> Уильям Джеймс был братом писателя Генри Джеймса. Говорили, что Генри пишет свои романы так, как если бы они были трудами по психологии, а Уильям сочиняет свои труды по психологии так, как если бы это были романы.

научного направления, как **когнитология**. Существует такое мнение, что зарождение этого направления произошло на одном из семинаров в Массачусетском технологическом институте в сентябре 1956 года. (Ниже показано, что это событие произошло всего лишь через два месяца после проведения конференции, на которой “родился” сам искусственный интеллект.) На этом семинаре Джордж Миллер представил доклад *The Magic Number Seven*, Ноам Хомский прочитал доклад *Three Models of Language*, а Аллен Ньюэлл и Герберт Саймон представили свою работу *The Logic Theory Machine*. В этих трех работах, получивших широкую известность, было показано, как можно использовать компьютерные модели для решения задач в области психологии, запоминания, обработки естественного языка и логического мышления. В настоящее время среди психологов находят широкое признание взгляды на то, что “любая теория познания должна напоминать компьютерную программу” [30], т.е. она должна подробно описывать механизм обработки информации, с помощью которого может быть реализована некоторая познавательная функция.

### Вычислительная техника (период с 1940 года по настоящее время)

- Каким образом можно создать эффективный компьютер?

Для успешного создания искусственного интеллекта требуется, во-первых, интеллект и, во-вторых, артефакт. Наиболее предпочтительным артефактом в этой области всегда был компьютер. Современный цифровой электронный компьютер был изобретен независимо и почти одновременно учеными трех стран, участвующих во Второй мировой войне. Первым операционным компьютером было электромеханическое устройство Heath Robinson<sup>12</sup>, созданное в 1940 году группой Алана Тьюринга для единственной цели — расшифровки сообщений, передаваемых немецкими войсками. В 1943 году та же группа разработала мощный компьютер общего назначения, получивший название Colossus, в конструкции которого применялись электронные лампы<sup>13</sup>. Первым операционным программируемым компьютером был компьютер Z-3, изобретенный Конрадом Цузе в Германии в 1941 году. Цузе изобрел также числа с плавающей точкой и создал первый язык программирования высокого уровня, Plankalkül. Первый электронный компьютер, ABC, был собран Джоном Атанасовым и его студентом Клиффордом Берри в период с 1940 по 1942 год в университете штата Айова. Исследования Атанасова почти не получили поддержки или признания; как оказалось, наибольшее влияние на развитие современных компьютеров оказал компьютер ENIAC, разработанный в составе секретного военного проекта в Пенсильванском университете группой специалистов, в состав которой входили Джон Мочли и Джон Экерт.

За прошедшее с тех пор полстолетие появилось несколько поколений компьютерного аппаратного обеспечения, причем каждое из них характеризовалось увеличением скорости и производительности, а также снижением цены. Производительность компьютеров, созданных на основе кремниевых микросхем, удваивается примерно через каждые 18 месяцев, и такая скорость роста наблюдается уже в течение

<sup>12</sup> Хет Робинсон (Heath Robinson), в честь которого названо это устройство, был карикатуристом, знаменитым тем, что изображал причудливые и абсурдно усложненные картины таких повседневных действий, как намазывание тостов маслом.

<sup>13</sup> В послевоенный период Тьюринг высказал желание применить эти компьютеры для исследований в области искусственного интеллекта, например для разработки одной из первых шахматных программ [1521], но его усилия были заблокированы британским правительством.

двух десятилетий. После достижения пределов этого роста потребуется молекулярная инженерия или какая-то другая, новая технология.

Безусловно, вычислительные устройства существовали и до появления электронного компьютера. Одно из первых автоматизированных устройств, появившееся еще в XVII столетии, рассматривалось на с. 41. Первым программируемым устройством был ткацкий станок, изобретенный в 1805 году Жозефом Марией Жаккардом (1752–1834), в котором использовались перфокарты для хранения инструкций по плетению узоров ткани. В середине XIX столетия Чарльз Бэббидж (1792–1871) разработал две машины, но ни одну из них не успел закончить. Его “разностная машина”, которая показана на обложке данной книги, предназначалась для вычисления математических таблиц, используемых в инженерных и научных проектах. В дальнейшем эта машина была построена и ее работа продемонстрирована в 1991 году в лондонском Музее науки [1481]. Другой замысел Бэббиджа, проект “аналитической машины”, был гораздо более амбициозным: в этой машине предусмотрено использование адресуемой памяти, хранимых программ и условных переходов, и она была первым артефактом, способным выполнять универсальные вычисления. Коллега Бэббиджа Ада Лавлейс, дочь поэта Лорда Байрона, была, возможно, первым в мире программистом. (В ее честь назван язык программирования Ada.) Она писала программы для незаконченной аналитической машины и даже размышляла над тем, что эта машина сможет играть в шахматы или сочинять музыку.

Искусственный интеллект во многом обязан также тем направлениям компьютерных наук, которые касаются программного обеспечения, поскольку именно в рамках этих направлений создаются операционные системы, языки программирования и инструментальные средства, необходимые для написания современных программ (и статей о них). Но эта область научной деятельности является также одной из тех, где искусственный интеллект в полной мере возмещает свои долг: работы в области искусственного интеллекта стали источником многих идей, которые затем были воплощены в основных направлениях развития компьютерных наук, включая разделение времени, интерактивные интерпретаторы, персональные компьютеры с оконными интерфейсами и поддержкой позиционирующих устройств, применение среды ускоренной обработки, создание типов данных в виде связных списков, автоматическое управление памятью и ключевые концепции символического, функционального, динамического и объектно-ориентированного программирования.

### **Теория управления и кибернетика (период с 1948 года по настоящее время)**

- Каким образом артефакты могут работать под своим собственным управлением?

Первое самоуправляемое устройство было построено Ктесибием из Александрии (примерно в 250 году до н.э.); это были водяные часы с регулятором, который поддерживал поток воды, текущий через эти часы с постоянным, предсказуемым расходом. Это изобретение изменило представление о том, на что могут быть способны устройства, созданные человеком. До его появления считалось, что только живые существа способны модифицировать свое поведение в ответ на изменения в окружающей среде. К другим примерам саморегулирующихся систем управления с обратной связью относятся регулятор паровой машины, созданный Джеймсом Уаттом (1736–1819), и термостат, изобретенный Корнелисом Дреббелем (1572–1633), кото-

рый изобрел также подводную лодку. Математическая теория устойчивых систем с обратной связью была разработана в XIX веке.

Центральной фигурой в создании науки, которая теперь именуется **теорией управления**, был Норберт Винер (1894–1964). Винер был блестящим математиком, который совместно работал со многими учеными, включая Бертрана Рассела, под влиянием которых у него появился интерес к изучению биологических и механических систем управления и их связи с познанием. Как и Крэг (который также использовал системы управления в качестве психологических моделей), Винер и его коллеги Артуро Розенблют и Джуллан Бигелоу бросили вызов ортодоксальным бихевиористским взглядам [1306]. Они рассматривали целенаправленное поведение как обусловленное действием регуляторного механизма, пытающего минимизировать “ошибку” — различие между текущим и целевым состоянием. В конце 1940-х годов Винер совместно с Уорреном Мак-Каллоком, Уолтером Питтсом и Джоном фон Нейманом организовал ряд конференций, на которых рассматривались новые математические и вычислительные модели познания; эти конференции оказали большое влияние на взгляды многих других исследователей в области наук о поведении. Книга Винера *Cybernetics* [1589], в которой было впервые дано определение **кибернетики** как науки, стала бестселлером и убедила широкие круги общественности в том, что мечта о создании машин, обладающих искусственным интеллектом, воплотилась в реальность.

Предметом современной теории управления, особенно той ее ветви, которая получила название *стохастического оптимального управления*, является проектирование систем, которые максимизируют **целевую функцию** во времени. Это примерно соответствует представлению авторов настоящей книги об искусственном интеллекте как о проектировании систем, которые действуют оптимальным образом. Почему же в таком случае искусственный интеллект и теория управления рассматриваются как две разные научные области, особенно если учесть, какие тесные взаимоотношения связывали их основателей? Ответ на этот вопрос состоит в том, что существует также тесная связь между математическими методами, которые были знакомы участникам этих разработок, и соответствующими множествами задач, которые были охвачены в каждом из этих подходов к описанию мира. Дифференциальное и интегральное исчисление, а также алгебра матриц, являющиеся инструментами теории управления, в наибольшей степени подходят для анализа систем, которые могут быть описаны с помощью фиксированных множеств непрерывно изменяющихся переменных; более того, точный анализ, как правило, осуществим только для линейных систем. Искусственный интеллект был отчасти основан как способ избежать ограничений математических средств, применявшихся в теории управления в 1950-х годах. Такие инструменты, как логический вывод и вычисления, позволили исследователям искусственного интеллекта успешно рассматривать некоторые проблемы (например, понимание естественного языка, зрение и планирование), полностью выходящие за рамки исследований, предпринимавшихся теоретиками управления.

### Лингвистика (период с 1957 года по настоящее время)

- Каким образом язык связан с мышлением?

В 1957 году Б.Ф. Скиннер опубликовал свою книгу *Verbal Behavior*. Это был всеобъемлющий, подробный отчет о результатах исследований по изучению языка, проведенных в рамках бихевиористского подхода, который был написан наиболее

выдающимся экспертом в этой области. Но весьма любопытно то, что рецензия к этой книге стала не менее известной, чем сама книга, и послужила причиной почти полного исчезновения интереса к бихевиоризму. Автором этой рецензии был Ноам Хомский, который сам только что опубликовал книгу с изложением своей собственной теории, *Syntactic Structures*. Хомский показал, что бихевиористская теория не позволяет понять истоки творческой деятельности, осуществляющейся с помощью языка, — она не объясняет, почему ребенок способен понимать и складывать предложения, которые он до сих пор никогда еще не слышал. Теория Хомского, основанная на синтаксических моделях, восходящих к работам древнеиндийского лингвиста Панини (примерно 350 год до н.э.), позволяла объяснить этот феномен, и, в отличие от предыдущих теорий, оказалась достаточно формальной для того, чтобы ее можно было реализовать в виде программ.

Таким образом, современная лингвистика и искусственный интеллект, которые “родились” примерно в одно и то же время и продолжают вместе расти, пересекаются в гибридной области, называемой **вычислительной лингвистикой** или **обработкой естественного языка**. Вскоре было обнаружено, что проблема понимания языка является гораздо более сложной, чем это казалось в 1957 году. Для понимания языка требуется понимание предмета и контекста речи, а не только анализ структуры предложений. Это утверждение теперь кажется очевидным, но сам данный факт не был широко признан до 1960-х годов. Основная часть ранних работ в области **представления знаний** (науки о том, как преобразовать знания в такую форму, с которой может оперировать компьютер) была привязана к языку и подпитывалась исследованиями в области лингвистики, которые, в свою очередь, основывались на результатах философского анализа языка, проводившегося в течение многих десятков лет.

---

### 1.3. ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

---

После ознакомления с изложенным выше материалом о предыстории искусственного интеллекта перейдем к изучению процесса развития самого искусственного интеллекта.

#### Появление предпосылок искусственного интеллекта (период с 1943 года по 1955 год)

Первая работа, которая теперь по общему признанию считается относящейся к искусственному интеллекту, была выполнена Уорреном Мак-Каллоком и Уолтером Питтсом [1017]. Они черпали вдохновение из трех источников: знание основ физиологии и назначения нейронов в мозгу; формальный анализ логики высказываний, взятый из работ Рассела и Уайтхеда; а также теория вычислений Тьюринга. Мак-Каллок и Питтс предложили модель, состоящую из искусственных нейронов, в которой каждый нейрон характеризовался как находящийся во “включенном” или “выключенном” состоянии, а переход во “включенное” состояние происходил в ответ на стимуляцию достаточного количества соседних нейронов. Состояние нейрона рассматривалось как “фактически эквивалентное высказыванию, в котором предлагается адекватное количество стимулов”. Работы этих ученых показали, например, что любая вычислимая функция может быть вычислена с по-

мошью некоторой сети из соединенных нейронов и что все логические связи (“И”, “ИЛИ”, “НЕ” и т.д.) могут быть реализованы с помощью простых сетевых структур. Кроме того, Мак-Каллок и Питтс выдвинули предположение, что сети, структурированные соответствующим образом, способны к обучению. Дональд Хебб [638] продемонстрировал простое правило обновления для модификации количества соединений между нейронами. Предложенное им правило, называемое теперь правилом **хеббовского обучения**, продолжает служить основой для моделей, широко используемых и в наши дни.

Два аспиранта факультета математики Принстонского университета, Марвин Минский и Дин Эдмондс, в 1951 году создали первый сетевой компьютер на основе нейронной сети. В этом компьютере, получившем название Snarc, использовалось 3000 электронных ламп и дополнительный механизм автопилота с бомбардировщика B-24 для моделирования сети из 40 нейронов. Аттестационная комиссия, перед которой Минский защищал диссертацию доктора философии, выразила сомнение в том, может ли работа такого рода рассматриваться как математическая, на что фон Нейман, по словам современников, возразил: “Сегодня — нет, но когда-то будет”. В дальнейшем Минский доказал очень важные теоремы, показывающие, с какими ограничениями должны столкнуться исследования в области нейронных сетей.

Кроме того, можно привести большое количество примеров других ранних работ, которые можно охарактеризовать как относящиеся к искусственному интеллекту, но именно Аллан Тьюринг впервые выразил полное представление об искусственном интеллекте в своей статье *Computing Machinery and Intelligence*, которая была опубликована в 1950 году. В этой статье он описал тест Тьюринга, принципы машинного обучения, генетические алгоритмы и обучение с подкреплением.

### Рождение искусственного интеллекта (1956 год)

В Принстонском университете проводил свои исследования еще один авторитетный специалист в области искусственного интеллекта, Джон Маккарти. После получения ученой степени Маккарти перешел в Дартмутский колледж, который и стал официальным местом рождения этой области знаний. Маккарти уговорил Марвина Минского, Клода Шеннона и Натаниэля Рочестера, чтобы они помогли ему собрать всех американских исследователей, проявляющих интерес к теории автоматов, нейронным сетям и исследованиям интеллекта. Они организовывали двухмесячный семинар в Дартмуте летом 1956 года. Всего на этом семинаре присутствовали 10 участников, включая Тренчарда Мура из Принстонского университета, Артура Самюэла из компании IBM, а также Рея Соломонова и Оливера Селфриджа из Массачусетского технологического института (Massachusetts Institute of Technology — MIT).

Два исследователя из технологического института Карнеги<sup>14</sup>, Аллен Ньюэлл и Герберт Саймон, буквально монополизировали все это представление. Тогда как другие могли лишь поделиться своими идеями и в некоторых случаях показать программы для таких конкретных приложений, как шашки, Ньюэлл и Саймон уже мог-

<sup>14</sup> Теперь это учебное заведение называется Университет Карнеги–Меллона (Carnegie–Mellon University — CMU).

ли продемонстрировать программу, проводящую рассуждения, Logic Theorist (LT)<sup>15</sup>, или логик-теоретик, в отношении которой Саймон заявил: “Мы изобрели компьютерную программу, способную мыслить в нечисловых терминах и поэтому решили поченную проблему о соотношении духа и тела”. Вскоре после этого семинара программа показала свою способность доказать большинство теорем из главы 2 труда Рассела и Уайтхеда *Principia Mathematica*. Сообщали, что Рассел пришел в восторг, когда Саймон показал ему, что эта программа предложила доказательство одной теоремы, более короткое, чем в *Principia*. Редакторы *Journal of Symbolic Logic* оказались менее подверженными эмоциям; они отказались принимать статью, в качестве соавторов которой были указаны Ньюэлл, Саймон и программа Logic Theorist.

Дартмутский семинар не привел к появлению каких-либо новых крупных открытий, но позволил познакомиться всем наиболее важным деятелям в этой научной области. Они, а также их студенты и коллеги из Массачусетского технологического института, Университета Карнеги-Меллона, Станфордского университета и компаний IBM занимали ведущее положение в этой области в течение следующих 20 лет. Возможно, больше всего сохранившимся результатом данного семинара было соглашение принять новое название для этой области, предложенное Маккарти, — **искусственный интеллект**. Возможно, лучше было бы назвать эту научную область “вычислительная рациональность”, но за ней закрепилось название “искусственный интеллект”.

Анализ предложений по тематике докладов для Дартмутского семинара [1014] позволяет понять, с чем связана необходимость преобразовать искусственный интеллект в отдельную область знаний. Почему нельзя было бы публиковать все работы, выполненные в рамках искусственного интеллекта, под флагом теории управления, или исследования операций, или теории решений, которые в конечном итоге имеют цели, аналогичные профессиональному интеллекту? Или почему искусственный интеллект не рассматривается как область математики? Ответом на эти вопросы, во-первых, является то, что искусственный интеллект с самого начала впитал идею моделирования таких человеческих качеств, как творчество, самосовершенствование и использование естественного языка. Эти задачи не рассматриваются ни в одной из указанных областей. Во-вторых, еще одним ответом является методология. Искусственный интеллект — это единственная из перечисленных выше областей, которая, безусловно, является одним из направлений компьютерных наук (хотя в исследовании операций также придается большое значение компьютерному моделированию), кроме того, искусственный интеллект — это единственная область, в которой предпринимаются попытки создания машин, действующих автономно в сложной, изменяющейся среде.

### **Ранний энтузиазм, большие ожидания (период с 1952 года по 1969 год)**

Первые годы развития искусственного интеллекта были полны успехов, хотя и достаточно скромных. Если учесть, какими примитивными были в то время компьютеры и инструментальные средства программирования, и тот факт, что лишь за несколько лет до этого компьютеры рассматривались как устройства, способные вы-

<sup>15</sup> Для написания программы LT Ньюэлл и Саймон разработали также язык обработки списков IPL. У них не было компилятора, поэтому эти ученые транслировали программы на своем языке в машинный код вручную. Чтобы избежать ошибок, они работали параллельно, называя друг другу двоичные числа после записи каждой команды, чтобы убедиться в том, что они совпадают.

полнять только арифметические, а не какие-либо иные действия, можно лишь удивляться тому, как удалось заставить компьютер выполнять операции, хоть немного напоминающие разумные. Интеллектуальное сообщество в своем большинстве продолжало считать, что “ни одна машина не сможет выполнить действие  $X$ ”. (Длинный список таких  $X$ , собранный Тьюрингом, приведен в главе 26.) Вполне естественно, что исследователи в области искусственного интеллекта отвечали на это, демонстрируя способность решать одну задачу  $X$  за другой. Джон Маккарти охарактеризовал этот период как эпоху восклицаний: “Гляди, мама, что я умею!”

За первыми успешными разработками Ньюэлла и Саймона последовало создание программы общего решателя задач (General Problem Solver — GPS). В отличие от программы Logic Theorist, эта программа с самого начала была предназначена для моделирования процедуры решения задач человеком. Как оказалось, в пределах того ограниченного класса головоломок, которые была способна решать эта программа, порядок, в котором она рассматривала подцели и возможные действия, был аналогичен тому подходу, который применяется людьми для решения таких же проблем. Поэтому программа GPS была, по-видимому, самой первой программой, в которой был воплощен подход к “организации мышления по такому же принципу, как и у человека”. Результаты успешного применения GPS и последующих программ в качестве модели познания позволили сформулировать знаменитую гипотезу **«физической символической системы»** ([1131]), в которой утверждается, что существует “физическая символическая система, которая имеет необходимые и достаточные средства для интеллектуальных действий общего вида”. Под этим подразумевается, что любая система, проявляющая интеллект (человек или машина), должна действовать по принципу манипулирования структурами данных, состоящими из символов. Ниже будет показано, что эта гипотеза во многих отношениях оказалась уязвимой для критики.

Работая в компании IBM, Натаниэль Рочестер и его коллеги создали некоторые из самых первых программ искусственного интеллекта. Герберт Гелернтер [532] сконструировал программу Geometry Theorem Prover (программа автоматического доказательства геометрических теорем), которая была способна доказывать такие теоремы, которые показались бы весьма сложными многим студентам-математикам. Начиная с 1952 года Артур Самюэл написал ряд программ для игры в шашки, которые в конечном итоге научились играть на уровне хорошо подготовленного любителя. В ходе этих исследований Самюэл опроверг утверждение, что компьютеры способны выполнять только то, чему их учили: одна из его программ быстро научилась играть лучше, чем ее создатель. Эта программа была продемонстрирована по телевидению в феврале 1956 года и произвела очень сильное впечатление на зрителей. Как и Тьюринг, Самюэл с трудом находил машинное время. Работая по ночам, он использовал компьютеры, которые все еще находились на испытательной площадке производственного предприятия компании IBM. Проблема ведения игр рассматривается в главе 6, а в главе 21 описаны и дополнены методы обучения, которые использовались Самюэлом.

Джон Маккарти перешел из Дартмутского университета в Массачусетский технологический институт и здесь в течение одного исторического 1958 года внес три крайне важных вклада в развитие искусственного интеллекта. В документе *MIT AI Lab Memo No. 1* Джон Маккарти привел определение нового языка высокого уровня **«Lisp»**, которому суждено было стать доминирующим языком программирования для искусственного интеллекта. Lisp остается одним из главных языков высокого

уровня, применяемых в настоящее время, будучи вместе с тем вторым по очередности появления языком такого типа, который был создан всего на один год позже чем Fortran. Разработав язык Lisp, Маккарти получил необходимый для него инструмент, но доступ к ограниченным и дорогостоящим компьютерным ресурсам продолжал оставаться серьезной проблемой. В связи с этим он совместно с другими сотрудниками Массачусетского технологического института изобрел режим разделения времени. В том же 1958 году Маккарти опубликовал статью под названием *Programs with Common Sense*, в которой он описал гипотетическую программу Advice Taker, которая может рассматриваться как первая полная система искусственного интеллекта. Как и программы Logic Theorist и Geometry Theorem Prover, данная программа Маккарти была предназначена для использования знаний при поиске решений задач. Но в отличие от других программ она была предназначена для включения общих знаний о мире. Например, Маккарти показал, что некоторые простые аксиомы позволяют этой программе разработать план оптимального маршрута автомобильной поездки в аэропорт, чтобы можно было успеть на самолет. Данная программа была также спроектирована таким образом, что могла принимать новые аксиомы в ходе обычной работы, а это позволяло ей приобретать компетентность в новых областях без перепрограммирования. Таким образом, в программе Advice Taker были воплощены центральные принципы представления знаний и проведения рассуждений, которые заключаются в том, что всегда полезно иметь формальное, явное представление о мире, а также о том, как действия агента влияют на этот мир и как приобрести способность манипулировать подобными представлениями с помощью дедуктивных процессов. Замечательной особенностью указанной статьи, которая вышла в 1958 году, является то, что значительная ее часть не потеряла своего значения и в наши дни.

Знаменитый 1958 год отмечен также тем, что именно в этот год Марвин Минский перешел в Массачусетский технологический институт. Но успешно складывавшееся на первых порах его сотрудничество с Маккарти продолжалось недолго. Маккарти настаивал на том, что нужно изучать способы представления и проведения рассуждений в формальной логике, тогда как Минский в большей степени интересовался тем, как довести программы до рабочего состояния, и в конечном итоге у него сформировалось отрицательное отношение к логике. В 1963 году Маккарти открыл лабораторию искусственного интеллекта в Станфордском университете. Разработанный им план использования логики для создания окончательной версии программы Advice Taker выполнялся еще быстрее, чем было задумано, благодаря открытию Дж.А. Робинсоном метода резолюции (полного алгоритма доказательства теорем для логики первого порядка; см. главу 9). Работы, выполненные в Станфордском университете, подчеркнули важность применения методов общего назначения для проведения логических рассуждений. В число логических приложений вошли системы формирования ответов на вопросы и планирования Корделла Грина [592], а также робототехнический проект Shakey, разрабатываемый в новом Станфордском научно-исследовательском институте (Stanford Research Institute — SRI). Последний проект, который подробно рассматривается в главе 25, впервые продемонстрировал полную интеграцию логических рассуждений и физической активности.

Минский руководил работой ряда студентов, выбравших для себя задачи ограниченных масштабов, для решения которых, как в то время казалось, требовалась интеллектуальность. Эти ограниченные проблемные области получили название  **микромиров**.

Программа Saint Джеймса Слэгга [1426] оказалась способной решать задачи интеграции в исчислении замкнутой формы, типичные для первых курсов колледжей. Программа Analogy Тома Эванса [448] решала задачи выявления геометрических аналогий, применяемые при проверке показателя интеллекта, аналогичные приведенной на рис. 1.2. Программа Student Дэниэла Боброва [142] решала изложенные в виде рассказа алгебраические задачи, подобные приведенной ниже.

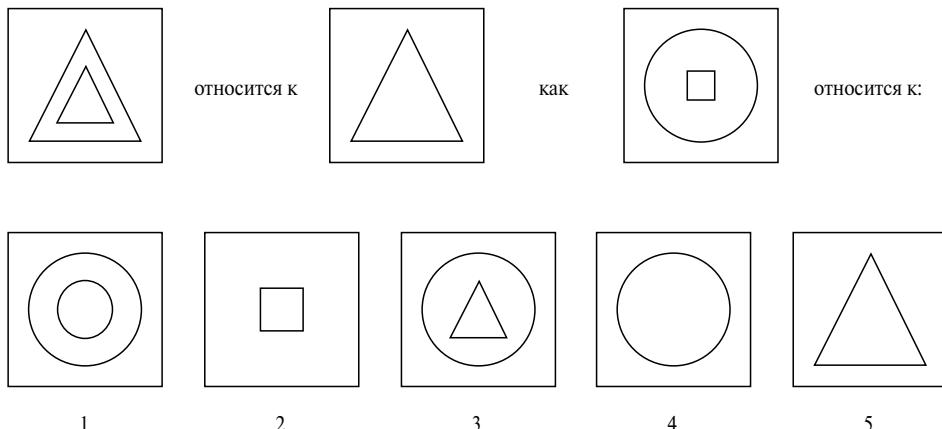
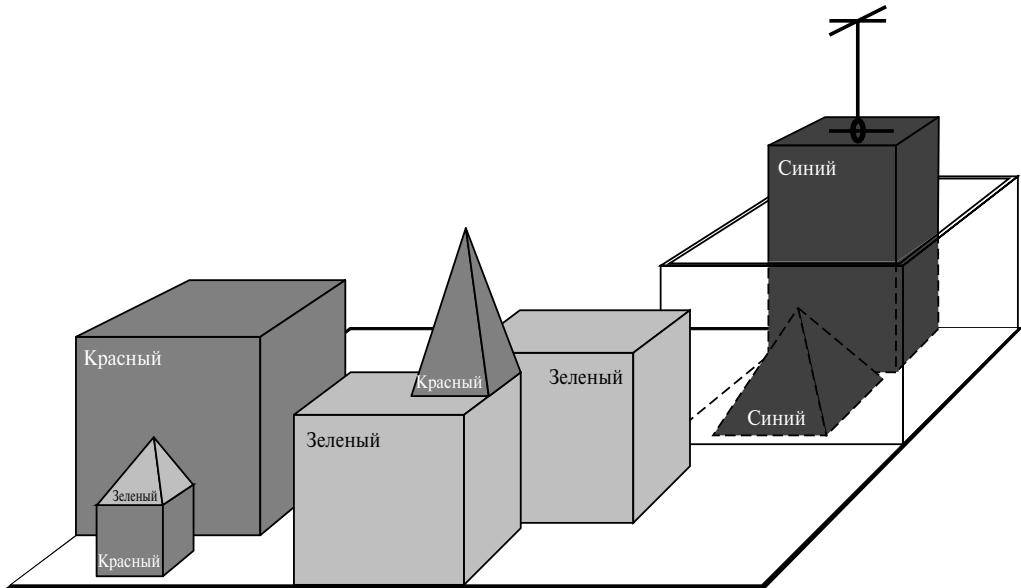


Рис. 1.2. Пример задачи, решаемой программой Analogy Эванса

Если количество заказов, полученных Томом, вдвое превышает квадратный корень из 20% опубликованных им рекламных объявлений, а количество этих рекламных объявлений равно 45, то каково количество заказов, полученных Томом?

Наиболее известным примером микромира был мир блоков, состоящий из множества цельных блоков, размещенных на поверхности стола (или, что более часто, на имитации стола), как показано на рис. 1.3. Типичной задачей в этом мире является изменение расположения блоков определенным образом с использованием манипулятора робота, который может захватывать по одному блоку одновременно. Мир блоков стал основой для проекта системы технического зрения Дэвида Хаффмена [702], работы по изучению зрения и распространения (удовлетворения) ограничений Дэвида Уолтса [1552], теории обучения Патрика Уинстона [1602], программы понимания естественного языка Тэрри Винограда [1601] и планировщика в мире блоков Скотта Фалмана [450].

Бурно продвигались также исследования, основанные на ранних работах по созданию нейронных сетей Мак-Каллока и Питтса. В работе Винограда и Коуэна [1600] было показано, как нужно представить отдельную концепцию с помощью коллекции, состоящей из большого количества элементов, соответственно увеличивая надежность и степень распараллеливания их работы. Методы обучения Хебба были усовершенствованы в работах Берни Видроу [1587], [1586], который называл свои сети **адалинами**, а также Френка Розенблатта [1304], создателя **перцептронов**. Розенблатт доказал **теорему сходимости перцептрана**, которая подтверждает, что предложенный им алгоритм обучения позволяет корректировать количество соединений перцептрана в соответствии с любыми входными данными, при условии, что такое соответствие существует. Эта тема рассматривается в главе 20.



*Рис. 1.3. Сцена из мира блоков. Программа Shrdlu [1601] только что завершила выполнение команды “Найти блок, более высокий по сравнению с тем, который находится в манипуляторе, и поместить его в ящик”*

### Столкновение с реальностью (период с 1966 года по 1973 год)

С самого начала исследователи искусственного интеллекта не отличались спорожнностью, высказывая прогнозы в отношении своих будущих успехов. Например, часто цитировалось приведенное ниже предсказание Герберта Саймона, опубликованное им в 1957 году.

Я не ставлю перед собой задачу удивить или шокировать вас, но проще всего я могу подвести итог, сказав, что теперь мы живем в таком мире, где машины могут думать, учиться и создавать. Более того, их способность выполнять эти действия будет продолжать расти до тех пор, пока (в обозримом будущем) круг проблем, с которыми смогут справиться машины, будет сопоставим с тем кругом проблем, где до сих пор был нужен человеческий мозг.

Такие выражения, как “обозримое будущее”, могут интерпретироваться по-разному, но Саймон сделал также более конкретный прогноз, что через десять лет компьютер станет чемпионом мира по шахматам и что машиной будут доказаны все важные математические теоремы. Эти предсказания сбылись (или почти сбылись) не через десять лет, а через сорок. Чрезмерный оптимизм Саймона был обусловлен тем, что первые системы искусственного интеллекта демонстрировали многообещающую производительность, хотя и на простых примерах. Но почти во всех случаях эти ранние системы терпели сокрушительное поражение, сталкиваясь с более широким кругом проблем или с более трудными проблемами.

Сложности первого рода были связаны с тем, что основная часть ранних программ не содержала знаний или имела лишь небольшой объем знаний о своей предметной области; их временные успехи достигались за счет простых синтаксических манипуляций. Типичная для этого периода история произошла при проведении

первых работ по машинному переводу текста на естественном языке, которые щедро финансировались Национальным научно-исследовательским советом США (U.S. National Research Council) в попытке ускорить перевод советских научных статей во время того периода бурной деятельности, который начался вслед за запуском в СССР первого искусственного спутника Земли в 1957 году. Вначале считалось, что для сохранения точного смысла предложений достаточно провести простые синтаксические преобразования, основанные на грамматиках русского и английского языков, и замену слов с использованием электронного словаря. Но дело в том, что для устранения неоднозначности и определения смысла предложения в процессе перевода необходимо обладать общими знаниями о предметной области. Возникающие при этом сложности иллюстрируются знаменитым обратным переводом фразы “the spirit is willing but the flesh is weak” (дух полон желаний, но плоть слаба), в результате которого получилось следующее: “the vodka is good but the meat is rotten” (водка хороша, но мясо испорчено). В 1966 году в отчете одного консультативного комитета было отмечено, что “машиинный перевод научного текста общего характера не осуществлен и не будет осуществлен в ближайшей перспективе”. Все финансирование академических проектов машинного перевода правительством США было свернуто. В настоящее время машинный перевод является несовершенным, но широко применяемым инструментальным средством обработки технических, коммерческих, правительственные документов, а также документов, опубликованных в Internet.

Сложности второго рода были связаны с неразрешимостью многих проблем, решение которых пытались найти с помощью искусственного интеллекта. В большинстве ранних программ искусственного интеллекта решение задач осуществлялось по принципу проверки различных комбинаций возможных шагов, которая проводилась до тех пор, пока не будет найдено решение. На первых порах такая стратегия приводила к успеху, поскольку микромиры содержали очень небольшое количество объектов, поэтому предусматривали лишь незначительный перечень возможных действий и позволяли находить очень короткие последовательности решения. До того как была разработана теория вычислительной сложности, было широко распространено такое мнение, что для “масштабирования” задач до уровня более крупных проблем достаточно просто применить более быстродействующие аппаратные средства с большим объемом памяти. Например, оптимизм, с которым были встречены сообщения о разработке метода доказательства теорем с помощью резолюции, быстро угас, когда исследователи не смогли доказать таким образом теоремы, которые включали чуть больше нескольких десятков фактов. Как оказалось, *то, что программа может найти решение в принципе, не означает, что эта программа действительно содержит все механизмы, позволяющие найти данное решение на практике.*

Иллюзия неограниченной вычислительной мощи распространялась не только на программы решения задач. Ранние эксперименты в области ~~эволюции~~ машин (которая теперь известна под названием **разработка генетических алгоритмов**) [502], [503] были основаны на уверенности в том, что внесение соответствующего ряда небольших изменений в машинный код программы позволяет создать программу решения любой конкретной простой задачи, обладающую высокой производительностью. Безусловно, что сам этот подход является вполне обоснованным. Поэтому общая идея состояла в том, что необходимо проверять случайные мутации (изменения в коде) с помощью процесса отбора для сохранения мутаций, которые кажутся полезными. На эти эксперименты было потрачено тысячи часов процессорного време-

ни, но никаких признаков прогресса не было обнаружено. В современных генетических алгоритмах используются лучшие способы представления, которые показывают более успешные результаты.

Одним из основных критических замечаний в адрес искусственного интеллекта, содержащихся в отчете Лайтхилла [930], который лег в основу решения британского правительства прекратить поддержку исследований в области искусственного интеллекта во всех университетах, кроме двух, была неспособность справиться с “комбинаторным взрывом” — стремительным увеличением сложности задачи. (Это — официальная версия событий, а в устном изложении рисуется немного иная и более красочная картина, в которой проявляются политические амбиции и личные интересы, описание которых выходит за рамки данного изложения.)

Сложности третьего рода возникли в связи с некоторыми фундаментальными ограничениями базовых структур, которые использовались для выработки интеллектуального поведения. Например, в книге Минского и Пейпера *Perceptrons* [1054] было доказано, что перцептроны (простая форма нейронной сети) могут продемонстрировать способность изучить все, что возможно представить с их помощью, но, к сожалению, они позволяют представить лишь очень немногое. В частности, перцептрон с двумя входами нельзя обучить распознаванию такой ситуации, при которой на два его входа подаются разные сигналы. Хотя полученные этими учеными результаты не распространяются на более сложные, многослойные сети, вскоре было обнаружено, что финансы, выделенные на поддержку исследований в области нейронных сетей, почти не приносят никакой отдачи. Любопытно отметить, что новые алгоритмы обучения путем обратного распространения для многослойных сетей, которые стали причиной возрождения необычайного интереса к исследованиям в области нейронных сетей в конце 1980-х годов, фактически были впервые открыты в 1969 году [201].

### **Системы, основанные на знаниях: могут ли они стать ключом к успеху (период с 1969 года по 1979 год)**

Основной подход к решению задач, сформированный в течение первого десятилетия исследований в области искусственного интеллекта, представлял собой механизм поиска общего назначения, с помощью которого предпринимались попытки связать в единую цепочку элементарные этапы проведения рассуждений для формирования полных решений. Подобные подходы получили название **слабых методов**, поскольку они не позволяли увеличить масштабы своего применения до уровня более крупных или более сложных экземпляров задач, несмотря на то, что были общими. Альтернативным по сравнению со слабыми методами стал подход, предусматривающий использование более содержательных знаний, относящихся к проблемной области, который позволяет создавать более длинные цепочки шагов логического вывода и дает возможность проще справиться с теми проблемными ситуациями, которые обычно возникают в специализированных областях знаний. Как известно, чтобы решить достаточно сложную задачу, необходимо уже почти полностью знать ответ.

Одним из первых примеров реализации такого подхода была программа Dendral [205]. Она была разработана в Станфордском университете группой ученых, в которую вошли Эд Фейгенбаум (бывший студент Герберта Саймона), Брюс Бьюкенен

(философ, который сменил специальность и стал заниматься компьютерными науками) и Джошуа Ледерберг (лауреат Нобелевской премии в области генетики). Эта группа занималась решением проблемы определения структуры молекул на основе информации, полученной от масс-спектрометра. Вход этой программы состоял из химической формулы соединения (например,  $C_6H_{13}NO_2$ ) и спектра масс, позволяющего определять массы различных фрагментов молекулы, который формировался при бомбардировке молекулы потоком электронов. Например, спектр масс может содержать пик в точке  $m=15$ , соответствующий массе метилового фрагмента ( $CH_3$ ).

Первая, примитивная версия этой программы предусматривала выработку всех возможных структур, совместимых с данной формулой, после чего предсказывала, какой спектр масс должен наблюдаться для каждой из этих структур, сравнивая его с фактическим спектром. Вполне можно ожидать, что такая задача применительно к молекулам более крупных размеров становится неразрешимой. Поэтому разработчики программы Dendral проконсультировались с химиками-аналитиками и пришли к выводу, что следует попытаться организовать работу по принципу поиска широко известных картин расположения пиков в спектре, которые указывают на наличие общих подструктур в молекуле. Например, для распознавания кетоновых подгрупп ( $C=O$ ) с атомными весами 28 может использоваться приведенное ниже правило.

**if** имеются два пика в точках  $x_1$  и  $x_2$ , такие, что:

- a)  $x_1 + x_2 = M + 28$  (где  $M$  — масса всей молекулы);
- б) в точке  $x_1 - 28$  — высокий пик;
- в) в точке  $x_2 - 28$  — высокий пик;
- г) по меньшей мере в одной из точек  $x_1$  и  $x_2$  — высокий пик,

**then** существует кетоновая подгруппа.

Применение способа, предусматривающего распознавание того, что молекула содержит какие-то конкретные подструктуры, позволило весьма значительно сократить количество возможных кандидатов, подлежащих проверке. В конечном итоге программа Dendral оказалась очень мощной, и причины этого описаны ниже.

Все относящиеся к делу теоретические знания, требуемые для решения указанных проблем, были преобразованы в [компоненте предсказания спектра] из наиболее общей формы (из “исходных принципов”) в эффективные специальные формы (в “рецепты поваренной книги”) [458].

Значение программы Dendral состояло в том, что это была первая успешно созданная экспертная система, основанная на широком использовании знаний: ее способность справляться с поставленными задачами была обусловлена применением большого количества правил специального назначения. В более поздних системах также широко применялся основной принцип подхода, реализованного Маккарти в программе Advice Taker, — четкое отделение знаний (в форме правил) от компонента, обеспечивающего проведение рассуждений.

Руководствуясь этим опытом, Фейгенбаум и другие специалисты из Станфордского университета приступили к разработке проекта эвристического программирования (Heuristic Programming Project — HPP), целью которого было исследование того, в какой степени созданная ими новая методология **экспертных систем** может быть применена в других областях интеллектуальной деятельности человека. На очередном этапе основные усилия были сосредоточены в области медицинской диагностики. Фейгенбаум, Бьюкенен и доктор Эдвард Шортлифф разработали

программу Mycin для диагностики инфекционных заболеваний кровеносной системы. После ввода в нее примерно 450 правил программа Mycin приобрела способность работать на уровне некоторых экспертов, а также показала значительно более лучшие результаты по сравнению с врачами, имеющими не такой большой стаж. Она также обладала двумя важными отличительными особенностями по сравнению с программой Dendral. Во-первых, в отличие от правил Dendral не существовала общая теоретическая модель, на основании которой мог бы осуществляться логический вывод правил Mycin. Для выявления этих правил приходилось широко применять знания, полученные от экспертов, которые, в свою очередь, приобретали эти знания с помощью учебников, других экспертов и непосредственного опыта, накопленного путем изучения практических случаев. Во-вторых, в этих правилах приходилось учитывать ту степень неопределенности, которой характеризуются знания в области медицины. В программе Mycin применялось исчисление неопределенностей на основе так называемых **коэффициентов уверенности** (см. главу 13), которое (в то время) казалось вполне соответствующим тому, как врачи оценивают влияние объективных данных на диагноз.

Важность использования знаний в проблемной области стала также очевидной и для специалистов, которые занимались проблемами понимания естественного языка. Хотя система понимания естественного языка Shrdlu, разработанная Тэрри Виноградом, стала в свое время предметом всеобщего восхищения, ее зависимость от результатов синтаксического анализа вызвала появление примерно таких же проблем, которые обнаружились в ранних работах по машинному переводу. Эта система была способна преодолеть неоднозначность и правильно понимала ссылки, выраженные с помощью местоимений, но это в основном было связано с тем, что она специально предназначалась только для одной области — для мира блоков. Некоторые исследователи, включая Юджина Чарняка, коллегу и аспиранта Винограда в Массачусетском технологическом институте, указывали, что для обеспечения надежного понимания языка потребуются общие знания о мире и общий метод использования этих знаний.

Работавший в Йельском университете Роджер Шенк, лингвист, ставший исследователем в области искусственного интеллекта, еще более ярко выразил эту мысль, заявив, что “такого понятия, как синтаксис, не существует”. Это заявление вызвало возмущение многих лингвистов, но послужило началом полезной дискуссии. Шенк со своими студентами создал ряд интересных программ [425], [1358], [1359], [1590]. Задача всех этих программ состояла в обеспечении понимания естественного языка. Но в них основной акцент был сделан в меньшей степени на языке как таковом и в большей степени на проблемах представления и формирования рассуждений с помощью знаний, требуемых для понимания языка. В число рассматриваемых проблем входило представление стереотипных ситуаций [314], описание организации человеческой памяти [829], [1287], а также понимание планов и целей [1591].

В связи с широким ростом количества приложений, предназначенных для решения проблем реального мира, столь же широко возрастили потребности в создании работоспособных схем представления знаний. Было разработано большое количество различных языков для представления знаний и проведения рассуждений. Некоторые из них были основаны на логике, например, в Европе получил распространение язык Prolog, а в Соединенных Штатах широко применялось семейство языков Planner. В других языках, основанных на выдвинутой Минским идеей  фреймов

[1053], был принят более структурированный подход, предусматривающий сбор фактов о конкретных типах объектов и событий, а также упорядочение этих типов в виде крупной таксономической иерархии, аналогичной биологической таксономии.

### **Превращение искусственного интеллекта в индустрию (период с 1980 года по настоящее время)**

Первая успешно действующая коммерческая экспертная система, R1, появилась в компании DEC (Digital Equipment Corporation) [1026]. Эта программа помогала составлять конфигурации для выполнения заказов на новые компьютерные системы; к 1986 году она позволяла компании DEC экономить примерно 40 миллионов долларов в год. К 1988 году группой искусственного интеллекта компании DEC было развернуто 40 экспертных систем, а в планах дальнейшего развертывания было предусмотрено еще большее количество таких систем. В компании Du Pont применялось 100 систем, в разработке находилось еще 500, а достигнутая экономия составляла примерно 10 миллионов долларов в год. Почти в каждой крупной корпорации США была создана собственная группа искусственного интеллекта и либо применялись экспертные системы, либо проводились их исследования.

В 1981 году в Японии было объявлено о развертывании проекта создания компьютера “пятого поколения” — 10-летнего плана по разработке интеллектуальных компьютеров, работающих под управлением языка Prolog. В ответ на это в Соединенных Штатах была сформирована корпорация Microelectronics and Computer Technology Corporation (MCC) как научно-исследовательский консорциум, предназначенный для обеспечения конкурентоспособности американской промышленности. И в том и в другом случае искусственный интеллект стал частью общего плана, включая его применение для проектирования микросхем и проведения исследований в области человеко-машинного интерфейса. Но амбициозные цели, поставленные перед специалистами в области искусственного интеллекта в проектах MCC и компьютеров пятого поколения, так и не были достигнуты. Тем не менее в Британии был выпущен отчет Олви (Alvey)<sup>16</sup>, в котором предусматривалось возобновление финансирования, урезанного на основании отчета Лайтхилла.

В целом в индустрии искусственного интеллекта произошел бурный рост, начиная с нескольких миллионов долларов в 1980 году и заканчивая миллиардами долларов в 1988 году. Однако вскоре после этого наступил период, получивший название “зимы искусственного интеллекта”, в течение которого пострадали многие компании, поскольку не сумели выполнить своих заманчивых обещаний.

### **Возвращение к нейронным сетям (период с 1986 года по настоящее время)**

Хотя основная часть специалистов по компьютерным наукам прекратила исследования в области нейронных сетей в конце 1970-х годов, работу в этой области продолжили специалисты из других научных направлений. Такие физики, как Джон Хопфилд [674], использовали методы из статистической механики для анализа

<sup>16</sup> Чтобы не ставить себя в затруднительное положение, авторы этого отчета изобрели новую научную область, получившую название “интеллектуальные системы, основанные на знаниях” (Intelligent Knowledge-Based Systems — IKBS), поскольку термин “искусственный интеллект” был уже официально отменен.

свойств хранения данных и оптимизации сетей, рассматривая коллекции узлов как коллекции атомов. Психологи, включая Дэвида Румельхарта и Джефа Хинтона, продолжали исследовать модели памяти на основе нейронных сетей. Как будет описано в главе 20, настоящий прорыв произошел в середине 1980-х годов, когда по меньшей мере четыре разные группы снова открыли алгоритм обучения путем обратного распространения, впервые предложенный в 1969 году Брайсоном и Хо [201]. Этот алгоритм был применен для решения многих проблем обучения в компьютерных науках и психологии, а после публикации результатов его использования в сборнике статей *Parallel Distributed Processing* [1318] всеобщее внимание привлек тот факт, насколько разнообразными оказались области его применения.

Эти так называемые **коннекционистские** (основанные на соединениях) модели интеллектуальных систем многими рассматривались как непосредственно конкурирующие и с символическими моделями, разрабатываемыми Ньюэллом и Саймоном, и с логицистским подходом, предложенным Маккарти и другими [1442]. По-видимому, не следует отрицать, что на некотором уровне мышления люди манипулируют символами; и действительно, в книге Терренса Дикона под названием *The Symbolic Species* [354] указано, что способность манипулировать символами — определяющая характеристика человека, но наиболее горячие сторонники коннекционизма поставили под сомнение то, что на основании манипулирования символами действительно можно полностью объяснить какие-то познавательные процессы в подробных моделях познания. Вопрос остается открытым, но современный взгляд на эту проблему состоит в том, что коннекционистский и символический подходы являются взаимодополняющими, а не конкурирующими.

### **Превращение искусственного интеллекта в науку (период с 1987 года по настоящее время)**

В последние годы произошла буквально революция как в содержании, так и в методологии работ в области искусственного интеллекта<sup>17</sup>. В настоящее время гораздо чаще встречаются работы, которые основаны на существующих теориях, а не содержат описания принципиально новых открытий; утверждения, изложенные в этих работах, основаны на строгих теоремах или надежных экспериментальных свидетельствах, а не на интуиции; при этом обоснованность сделанных выводов подтверждается на реальных практических приложениях, а не на игрушечных примерах.

Появление искусственного интеллекта отчасти стало результатом усилий по преодолению ограничений таких существующих научных областей, как теория управления и статистика, но теперь искусственный интеллект включил в себя и эти области. В одной из своих работ Дэвид Макаллестер [1006] выразил эту мысль следующим образом.

В ранний период развития искусственного интеллекта казалось вероятным, что в результате появления новых форм символьических вычислений, например фреймов и семантиче-

<sup>17</sup> Некоторые охарактеризовали эту смену подходов как победу **теоретиков** (тех, кто считает, что теории искусственного интеллекта должны быть основаны на строгих математических принципах) над **экспериментаторами** (теми, кто предпочитает проверить множество идей, написать какие-то программы, а затем оценить те из них, которые кажутся работоспособными). Оба подхода являются важными. А смещение акцентов в пользу теоретической обоснованности свидетельствует о том, что данная область достигла определенного уровня стабильности и зрелости. Будет ли когда-либо такая стабильность нарушена новой идеей, родившейся в экспериментах, — это другой вопрос.

ских сетей, основная часть классической теории станет устаревшей. Это привело к определенной форме самоизоляции, характеризовавшейся тем, что искусственный интеллект в значительной степени отделился от остальной части компьютерных наук. В настоящее время такой изоляционизм преодолен. Появилось признание того, что машинное обучение не следует отделять от теории информации, что проведение рассуждений в условиях неопределенности нельзя изолировать от стохастического моделирования, что поиск не следует рассматривать отдельно от классической оптимизации и управления и что автоматизированное формирование рассуждений не должно трактоваться как независимое от формальных методов и статистического анализа.

С точки зрения методологии искусственный интеллект наконец-то твердо перешел на научные методы. Теперь, для того чтобы быть принятыми, гипотезы должны подвергаться проверке в строгих практических экспериментах, а значимость результатов должна подтверждаться данными статистического анализа [275]. Кроме того, в настоящее время имеется возможность воспроизводить эксперименты с помощью Internet, а также совместно используемых репозитариев тестовых данных и кода.

Именно по этому принципу развивается область распознавания речи. В 1970-е годы было опробовано широкое разнообразие различных архитектур и подходов. Многие из них оказались довольно надуманными и недолговечными и были продемонстрированы только на нескольких специально выбранных примерах. В последние годы доминирующее положение в этой области заняли подходы, основанные на использовании **скрытых марковских моделей** (Hidden Markov Model — HMM). Описанное выше современное состояние искусственного интеллекта подтверждается двумя особенностями моделей HMM. Во-первых, они основаны на строгой математической теории. Это позволяет исследователям речи использовать в своей работе математические результаты, накопленные в других областях за несколько десятилетий. Во-вторых, они получены в процессе обучения программ на крупном массиве реальных речевых данных. Это гарантирует обеспечение надежных показателей производительности, а в строгих слепых испытаниях модели HMM неизменно улучшают свои показатели. Технология распознавания речи и связанная с ней область распознавания рукописных символов уже совершают переход к созданию широко применяемых индустриальных и потребительских приложений.

Нейронные сети также следуют этой тенденции. Основная часть работ по нейронным сетям, осуществленных в 1980-х годах, была проведена в попытке оценить масштабы того, что должно быть сделано, а также понять, в чем нейронные сети отличаются от “традиционных” методов. В результате использования усовершенствованной методологии и теоретических основ исследователи в этой области достигли такого уровня понимания, что теперь нейронные сети стали сопоставимыми с соответствующими технологиями из области статистики, распознавания образов и машинного обучения, а наиболее перспективная методология может быть применена к каждому из этих приложений. В результате этих разработок была создана так называемая технология **анализа скрытых закономерностей в данных** (data mining), которая легла в основу новой, быстро растущей отрасли информационной индустрии.

Знакомство широких кругов специалистов с книгой Джуди Перла *Probabilistic Reasoning in Intelligent Systems* [1191] привело к признанию важности теории вероятностей и теории решений для искусственного интеллекта, что последовало за возрождением интереса к этой теме, вызванной статьей Питера Чизмана *In Defense of Probability* [242]. Для обеспечения эффективного представления неопределенных

знаний и проведения на их основе строгих рассуждений были разработаны формальные средства **байесовских сетей**. Этот подход позволил преодолеть многие проблемы систем вероятностных рассуждений, возникавшие в 1960–1970-х гг.; теперь он стал доминирующим в таких направлениях исследований искусственного интеллекта, как формирование рассуждений в условиях неопределенности и экспертные системы. Данный подход позволяет организовать обучение на основе опыта и сочетает в себе лучшие достижения классического искусственного интеллекта и нейронных сетей. В работах Джуди Перла [1186], а также Эрика Горвица и Дэвида Хекермана [688], [689] была развита идея *нормативных экспертных систем*. Таковыми являются системы, которые действуют рационально, в соответствии с законами теории решений, а не пытаются имитировать мыслительные этапы в работе людей-экспертов. Операционная система Windows<sup>TM</sup> включает несколько нормативных диагностических экспертных систем, применяемых для устранения нарушений в работе. Эта область рассматривается в главах 13–16.

Аналогичные бескровные революции произошли в области робототехники, компьютерного зрения и представления знаний. Благодаря лучшему пониманию исследовательских задач и свойств, обусловливающих их сложность, в сочетании с все-возрастающим усложнением математического аппарата, удалось добиться формирования реальных планов научных исследований и перейти к использованию более надежных методов. Но во многих случаях формализация и специализация привели также к фрагментации направлений, например, такие темы, как машинное зрение и робототехника, все больше отделяются от “основного направления” работ по искусственному интеллекту. Снова добиться объединения этих разрозненных областей можно на основе единого взгляда на искусственный интеллект как науку проектирования рациональных агентов.

### **Появление подхода, основанного на использовании интеллектуальных агентов (период с 1995 года по настоящее время)**

Вдохновленные успехами в решении указанных проблем искусственного интеллекта, исследователи также вновь приступили к решению проблемы “целостного агента”. Наиболее широко известным примером создания полной архитектуры агента является работа Аллена Ньюэлла, Джона Лэрда и Поля Розенблума [880], [1125] над проектом Soar. Для того чтобы проще было разобраться в работе агентов, внедренных в реальную среду с непрерывным потоком сенсорных входных данных, были применены так называемые *ситуационные движения*. Одним из наиболее важных примеров среди для интеллектуальных агентов может служить Internet. Системы искусственного интеллекта стали настолько распространенными в приложениях для Web, что суффикс “-бот” (сокращение от робот) вошел в повседневный язык. Более того, технологии искусственного интеллекта легли в основу многих инструментальных средств Internet, таких как машины поиска, системы, предназначенные для выработки рекомендаций, и системы создания Web-узлов.

Пересмотру с учетом нового представления о роли агентов было подвергнуто не только первое издание данной книги [1328], но и другие новейшие труды по этой теме [1146], [1227]. Одним из следствий попыток создания полных агентов стало понимание того, что ранее изолированные подобласти искусственного интеллекта могут потребовать определенной реорганизации, когда возникнет необходимость

снова связать воедино накопленные в них результаты. В частности, теперь широко признано, что сенсорные системы (системы машинного зрения, эхолокации, распознавания речи и т.д.) не способны предоставить абсолютно надежную информацию о среде. Поэтому системы проведения рассуждений и планирования должны быть приспособленными к работе в условиях неопределенности. Вторым важным следствием изменения взглядов на роль агентов является то, что исследования в области искусственного интеллекта теперь необходимо проводить в более тесном контакте с другими областями, такими как теория управления и экономика, которые также имеют дело с агентами.

## 1.4. СОВРЕМЕННОЕ СОСТОЯНИЕ РАЗРАБОТОК

Какие возможности предоставляет искусственный интеллект в наши дни? Краткий ответ на этот вопрос сформулировать сложно, поскольку в этом научном направлении существует слишком много подобластей, в которых выполняется очень много исследований. Ниже в качестве примеров перечислено лишь несколько приложений; другие будут указаны в следующих главах.

- **Автономное планирование и составление расписаний.** Работающая на удалении в сотни миллионов километров от Земли программа Remote Agent агентства NASA стала первой бортовой автономной программой планирования, предназначеннной для управления процессами составления расписания операций для космического аппарата [744]. Программа Remote Agent вырабатывала планы на основе целей высокого уровня, задаваемых с Земли, а также контролировала работу космического аппарата в ходе выполнения планов: обнаруживала, диагностировала и устраняла неполадки по мере их возникновения.
- **Ведение игр.** Программа Deep Blue компании IBM стала первой компьютерной программой, которой удалось победить чемпиона мира в шахматном матче, после того как она обыграла Гарри Каспарова со счетом 3,5:2,5 в показательном матче [577]. Каспаров заявил, что ощущал напротив себя за шахматной доской присутствие “интеллекта нового типа”. Журнал *Newsweek* описал этот матч под заголовком “Последний оборонительный рубеж мозга”. Стоимость акций IBM выросла на 18 миллиардов долларов.
- **Автономное управление.** Система компьютерного зрения Alvinn была обучена вождению автомобиля, придерживаясь определенной полосы движения. В университете CMU эта система была размещена в микроавтобусе, управляемом компьютером NavLab, и использовалось для проезда по Соединенным Штатам; на протяжении 2850 миль (4586,6 км) система обеспечивала рулевое управление автомобилем в течение 98% времени. Человек брал на себя управление лишь в течение остальных 2%, главным образом на выездных пандусах. Компьютер NavLab был оборудован видеокамерами, которые передавали изображения дороги в систему Alvinn, а затем эта система вычисляла наилучшее направление движения, основываясь на опыте, полученном в предыдущих учебных пробегах.
- **Диагностика.** Медицинские диагностические программы, основанные на вероятностном анализе, сумели достичь уровня опытного врача в нескольких

областях медицины. Хекерман [640] описал случай, когда ведущий специалист в области патологии лимфатических узлов не согласился с диагнозом программы в особо сложном случае. Создатели программы предложили, чтобы этот врач запросил у компьютера пояснения по поводу данного диагноза. Машина указала основные факторы, повлиявшие на ее решение, и объяснила нюансы взаимодействия нескольких симптомов, наблюдавшихся в данном случае. В конечном итоге эксперт согласился с решением программы.

- **Планирование снабжения.** Во время кризиса в Персидском заливе в 1991 году в армии США была развернута система DART (Dynamic Analysis and Replanning) [311] для обеспечения автоматизированного планирования поставок и составления графиков перевозок. Работа этой системы охватывала одновременно до 50 000 автомобилей, единиц груза и людей; в ней приходилось учитывать пункты отправления и назначения, маршруты, а также устранять конфликты между всеми параметрами. Методы планирования на основе искусственного интеллекта позволяли вырабатывать в течение считанных часов такие планы, для составления которых старыми методами потребовались бы недели. Представители агентства DARPA (Defense Advanced Research Project Agency — Управление перспективных исследовательских программ) заявили, что одно лишь это приложение сторицей окупило тридцатилетние инвестиции в искусственный интеллект, сделанные этим агентством.
- **Робототехника.** Многие хирурги теперь используют роботов-ассистентов в микрохирургии. Например, HipNav [398] — это система, в которой используются методы компьютерного зрения для создания трехмерной модели анатомии внутренних органов пациента, а затем применяется робототехническое управление для руководства процессом вставки протеза, заменяющего тазобедренный сустав.
- **Понимание естественного языка и решение задач.** Программа Proverb [938] — это компьютерная программа, которая решает кроссворды намного лучше, чем большинство людей; в ней используются ограничения, определяющие состав возможных заполнителей слов, большая база с данными о встречающихся ранее кроссвордах, а также множество различных источников информации, включая словари и оперативные базы данных, таких как списки кинофильмов и актеров, которые играли в этих фильмах. Например, эта программа способна определить, что одним из решений, подходящих для ключа “Nice Story”, является слово “ETAGE”, поскольку ее база данных содержит пару ключ–решение “Story in France/ETAGE”, а сама программа распознает, что шаблоны “Nice X” и “X in France” часто имеют одно и то же решение. Программа не знает, что Nice (Ницца) — город во Франции, но способна разгадать эту головоломку.

Выше приведено лишь несколько примеров систем искусственного интеллекта, которые существуют в настоящее время. Искусственный интеллект — это не магия и не научная фантастика, а сплав методов науки, техники и математики, вводное описание которых приведено в данной книге.

## 1.5. РЕЗЮМЕ

В настоящей главе дано определение искусственного интеллекта и описан исторический контекст, в котором развивалась эта область науки. Ниже приведены некоторые важные темы, которые рассматривались в этой главе.

- Взгляды ученых на искусственный интеллект не совпадают. Для того чтобы определить наиболее приемлемый для себя подход, необходимо ответить на два важных вопроса: “Интересует ли вас в основном мышление или поведение?” и “Стремитесь ли вы моделировать способности людей или строить свою работу исходя из идеального стандарта?”
- В данной книге принят подход, согласно которому интеллектуальность в основном связана с **рациональной деятельностью**. В идеальном случае **интеллектуальный агент** в любой ситуации предпринимает наилучшее возможное действие. В дальнейшем изложении рассматривается проблема создания агентов, которые являются интеллектуальными именно в этом смысле.
- Философы (начиная с 400 года до н.э.) заложили основы искусственного интеллекта, сформулировав идеи, что мозг в определенных отношениях напоминает машину, что он оперирует знаниями, закодированными на каком-то внутреннем языке, и что мышление может использоваться для выбора наилучших предпринимаемых действий.
- Математики предоставили инструментальные средства для манипулирования высказываниями, обладающими логической достоверностью, а также недостоверными вероятностными высказываниями. Кроме того, они заложили основу не только понимания того, что представляют собой вычисления, но и формирования рассуждений об алгоритмах.
- Экономисты formalизовали проблему принятия решений, максимизирующих ожидаемый выигрыш для лица, принимающего решение.
- Психологи подтвердили идею, что люди и животные могут рассматриваться как машины обработки информации. Лингвисты показали, что процессы использования естественного языка укладываются в эту модель.
- Компьютерные инженеры предоставили артефакты, благодаря которым стало возможным создание приложений искусственного интеллекта. Обычно программы искусственного интеллекта имеют большие размеры, и не могли бы работать без тех значительных достижений в повышении быстродействия и объема памяти, которые были достигнуты в компьютерной индустрии.
- Теория управления посвящена проектированию устройств, которые действуют оптимально на основе обратной связи со средой. Первоначально математические инструментальные средства теории управления весьма отличались от применяемых в искусственном интеллекте, но эти научные области все больше сближаются.
- История искусственного интеллекта характеризуется периодами успеха и неоправданного оптимизма, за которыми следовало снижение интереса и сокращение финансирования. В ней также были периоды, когда появлялись новые творческие подходы, а затем лучшие из них систематически совершенствовались.

- Искусственный интеллект развивался быстрее, чем обычно, в прошлое десятилетие, поскольку в этой области стали шире применяться научные методы экспериментирования и сравнения подходов.
- Последние достижения на пути понимания теоретических основ интеллектуальности неразрывно связаны с расширением возможностей реальных систем. Отдельные подобласти искусственного интеллекта стали в большей степени интегрированными, а сам искусственный интеллект успешно находит общую почву с другими научными дисциплинами.

## **БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ**

---

Состояние методологических основ искусственного интеллекта исследовано Гербертом Саймоном в его работе *The Sciences of the Artificial* [1417], в которой обсуждаются области научных исследований, относящиеся к сложным артефактам. В этой книге дано объяснение того, почему искусственный интеллект может рассматриваться как прикладная, и как теоретическая наука. В [275] дан краткий обзор методологии проведения экспериментов, применяемой в искусственном интеллекте. В [480] изложено определенное мнение о полезности теста Тьюринга, отражающее собственную оценку этого теста некоторыми учеными.

В книге Джона Хоглэнда *Artificial Intelligence: The Very Idea* [631] приведено интересное описание философских и практических проблем искусственного интеллекта. Когнитология хорошо описана в нескольких недавно опубликованных книгах [740], [1465], [1502] и *Encyclopedia of the Cognitive Sciences* [1599]. В [61] рассматривается синтаксическая часть современной лингвистики, в [249] предметом изложения является семантика, а в [756] описана компьютерная лингвистика.

Ранний период развития искусственного интеллекта описан в книге Фейгенбаума и Фельдмана *Computers and Thought* [459], в книге Минского *Semantic Information Processing* [1052] и в серии книг *Machine Intelligence*, изданной под редакцией Дональда Мичи. Большое количество важных статей было выпущено в виде антологий [964] и [1562]. Ранние работы по нейронным сетям собраны в работе *Neurocomputing* [29]. В книге *Encyclopedia of AI* [1397] содержатся обзорные статьи почти по каждой теме в искусственном интеллекте. Обычно эти статьи становятся хорошим введением перед ознакомлением с научно-исследовательской литературой по каждой теме.

Результаты новейших работ публикуются в трудах основных конференций по искусственному интеллекту: проводимой один раз в два года конференции *International Joint Conference on AI* (IJCAI), ежегодной конференции *European Conference on AI* (ECAI) и конференции *National Conference on AI*, более часто упоминаемой под названием AAAI (так сокращенно называется организация American Association for AI, под эгидой которой проводится эта конференция). Главными журналами по общим направлениям искусственного интеллекта являются *Artificial Intelligence*, *Computational Intelligence*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Intelligent Systems* и электронный *Journal of Artificial Intelligence Research*. Имеется также много конференций и журналов, посвященных определенным областям, которые будут указаны в соответствующих главах. Основными профессиональными обществами по искусственному интеллекту являются *American Association for Artificial*

*Intelligence (AAAI), ACM Special Interest Group in Artificial Intelligence (SIGART) и Society for Artificial Intelligence and Simulation of Behaviour (AISB).* В журнале *AI Magazine* организации AAAI можно найти много тематических и учебных статей, а на Web-узле этого общества ([aaai.org](http://aaai.org)) публикуются новости и основная информация.

## УПРАЖНЕНИЯ

Эти упражнения предназначены для организации на их основе творческой дискуссии, а некоторые из них могут быть определены как проекты с заданными сроками. Еще один вариант состоит в том, чтобы предпринять предварительные попытки их решения сейчас, а затем пересмотреть результаты этих попыток после завершения изучения книги.

- 1.1.** Самостоятельно сформулируйте определения следующих понятий: а) интеллектуальность; б) искусственный интеллект; в) агент.
- 1.2.** Прочитайте оригинальную статью Тьюринга по искусенному интеллекту [1520]. В этой статье он обсуждает несколько потенциальных возражений против предложенного им подхода и теста интеллектуальности. Какие из этих возражений все еще остаются весомыми в определенной степени? Действительно ли приведенные им опровержения этих возражений являются правильными? Можете ли вы выдвинуть новые возражения, которые следуют из событий, произошедших с тех пор, как Тьюринг написал свою статью? В этой статье он предсказал, что к 2000 году компьютер с вероятностью 30% будет успешно проходить пятиминутный тест Тьюринга с участием слабо подготовленного экспериментатора. Какие шансы, по вашему мнению, имел бы компьютер сегодня? Еще через 50 лет?
- 1.3.** Каждый год происходит вручение приза Лебнера (Loebner) создателям программы, которая показывает наилучшие результаты при прохождении определенной версии теста Тьюринга. Проведите исследование и сообщите о последнем победителе в соревновании за приз Лебнера. Какие методы используются в этой программе? Какой вклад внесла эта программа в развитие искусственного интеллекта?
- 1.4.** Существуют известные классы проблем, которые являются трудноразрешимыми для компьютеров, а в отношении других классов доказано, что они неразрешимы. Следует ли из этого вывод, что создание искусственного интеллекта невозможно?
- 1.5.** Предположим, что программа *Analogy* Эванса будет настолько усовершенствована, что сможет получать 200 очков при стандартной проверке показателя интеллекта. Означает ли это, что при этом будет создана программа, более интеллектуальная, чем человек? Обоснуйте свой ответ.
- 1.6.** Почему самоанализ (составление отчета о своих собственных сокровенных мыслях) может оказаться неточным? Как человек может оказаться неправ, обсуждая то, что он думает? Обоснуйте свой ответ.
- 1.7.** Изучите литературу по искусенному интеллекту, чтобы определить, могут ли следующие задачи в настоящее время быть решены компьютерами.

- а) Игра в настольный теннис (пинг-понг) на достаточно высоком уровне.
  - б) Вождение автомобиля в центре Каира.
  - в) Покупка в супермаркете недельного запаса продовольствия.
  - г) Покупка недельного запаса продовольствия в Web.
  - д) Участие в карточной игре бридж на конкурентоспособном уровне.
  - е) Открытие и доказательство новых математических теорем.
  - ж) Написание рассказа, который непременно должен быть смешным.
  - а) Предоставление компетентной юридической консультации в специализированной области законодательства.
  - и) Перевод в реальном времени разговорной речи с английского языка на шведский язык.
  - к) Выполнение сложной хирургической операции.
- В отношении задач, которые в настоящее время остаются неосуществимыми, попытайтесь узнать, в чем заключаются трудности, и предсказать, когда они будут преодолены (и произойдет ли это вообще).
- 1.8.** Некоторые авторы утверждают, что самой важной частью интеллекта служат сенсорные способности и моторные навыки и что “высокоуровневые” возможности неизбежно остаются паразитическими, поскольку являются простыми дополнениями к этим основным возможностям. И действительно, не подлежит сомнению, что развитие способностей к восприятию и моторных навыков происходило на протяжении почти всей эволюции, а их поддержка осуществляется в большей части мозга, тогда как искусственный интеллект сосредоточился на таких задачах, как ведение игры и формирование логического вывода, которые во многом оказались значительно более простыми по сравнению с восприятием и осуществлением действий в реальном мире. Не кажется ли вам, что традиционная направленность искусственного интеллекта на изучение высокоуровневых познавательных способностей не совсем оправдана?
- 1.9.** Почему результатом эволюции обычно становится появление систем, которые действуют рационально? Для достижения каких целей предназначены подобные системы?
- 1.10.** Являются ли рефлексорные действия (такие как отдергивание руки от горячей печи) рациональными? Являются ли они интеллектуальными?
- 1.11.** “Безусловно, компьютеры не могут быть интеллектуальными, ведь они способны выполнять только то, что диктуют им программисты”. Является ли последнее утверждение истинным и следует ли из него первое?
- 1.12.** “Безусловно, животные не могут быть интеллектуальными, ведь они способны выполнять только то, что диктуют им гены”. Является ли последнее утверждение истинным и следует ли из него первое?
- 1.13.** “Безусловно, животные, люди и компьютеры не могут быть интеллектуальными, ведь они способны выполнять только то, что диктуют атомам, из которых они состоят, законы физики”. Является ли последнее утверждение истинным и следует ли из него первое?

## 2 ИНТЕЛЛЕКТУАЛЬНЫЕ АГЕНТЫ

*В этой главе рассматриваются характеристики агентов, идеальных или неидеальных, разнообразие вариантов среды и вытекающая из этого классификация типов агентов.*

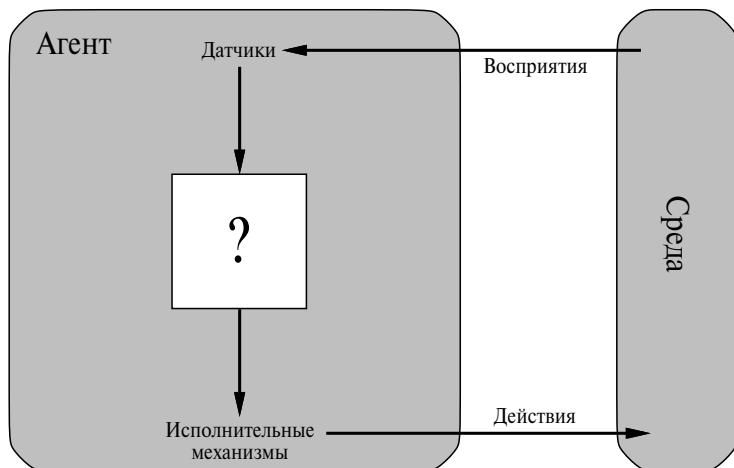
Как было указано в главе 1, понятие **рационального агента** является центральным в применяемом авторами данной книги подходе к искусственному интеллекту. В этой главе указанное понятие раскрывается более подробно. В ней показано, что концепция рациональности может применяться к самым различным агентам, действующим в любой среде, которую только можно себе представить. План авторов состоит в том, чтобы использовать эту концепцию в данной книге для разработки небольшого набора принципов проектирования для создания успешно действующих агентов — систем, которые вполне можно было бы назвать **интеллектуальными**.

Начнем с изучения агентов, вариантов среды и связей между ними. Наблюдая за тем, что некоторые агенты действуют лучше, чем другие, можно вполне обоснованно выдвинуть идею рационального агента; таковым является агент, который действует настолько успешно, насколько это возможно. Успехи, которых может добиться агент, зависят от характера среды; некоторые варианты среды являются более сложными, чем другие. В этой главе дана грубая классификация вариантов среды и показано, как свойства среды влияют на проектирование агентов, наиболее подходящих для данной среды. Здесь описан ряд основных, “скелетных” проектов агентов, которые будут облечены в плоть в остальной части книги.

### 2.1. АГЕНТЫ И ВАРИАНТЫ СРЕДЫ

Агентом является все, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов. Эта простая идея иллюстрируется на рис. 2.1. Человек, рассматриваемый в роли агента, имеет глаза, уши и другие органы чувств, а исполнительными механизмами для него служат руки, ноги, рот и другие части тела. Робот, выполняющий функции агента, в качестве датчиков может иметь видеокамеры и инфракрасные дальномеры, а его исполнительными механизмами могут являться различные двигатели. Программное обеспечение, выступающее в роли аген-

та, в качестве входных сенсорных данных получает коды нажатия клавиш, содержимое файлов и сетевые пакеты, а его воздействие на среду выражается в том, что программное обеспечение выводит данные на экран, записывает файлы и передает сетевые пакеты. Мы принимаем общее допущение, что каждый агент может воспринимать свои собственные действия (но не всегда их результаты).



*Рис. 2.1. Агент взаимодействует со средой с помощью датчиков и исполнительных механизмов*

Мы используем термин **восприятие** для обозначения полученных агентом сенсорных данных в любой конкретный момент времени. **Последовательностью актов восприятия** агента называется полная история всего, что было когда-либо воспринято агентом. Вообще говоря, **выбор агентом действия в любой конкретный момент времени может зависеть от всей последовательности актов восприятия, наблюдавшихся до этого момента времени**. Если существует возможность определить, какое действие будет выбрано агентом в ответ на любую возможную последовательность актов восприятия, то может быть дано более или менее точное определение агента. С точки зрения математики это равносильно утверждению, что поведение некоторого агента может быть описано с помощью **функции агента**, которая отображает любую конкретную последовательность актов восприятия на некоторое действие.

Может рассматриваться задача табуляции функции агента, которая описывает любого конкретного агента; для большинства агентов это была бы очень большая таблица (фактически бесконечная), если не устанавливается предел длины последовательностей актов восприятия, которые должны учитываться в таблице. Проводя эксперименты с некоторым агентом, такую таблицу в принципе можно сконструировать, проверяя все возможные последовательности актов восприятия и регистрируя, какие действия в ответ выполняет агент<sup>1</sup>. Такая таблица, безусловно, является внешним описанием агента. Внутреннее описание состоит в определении того, ка-

<sup>1</sup> Если агент для выбора своих действий использует определенную рандомизацию, то может потребоваться проверить каждую последовательность многократно, чтобы определить вероятность каждого действия. На первый взгляд кажется, что выбор действий случайным образом является довольно неразумным, но ниже в этой главе будет показано, что такая организация функционирования может оказаться весьма интеллектуальной.

кая функция агента для данного искусственного агента реализуется с помощью **программы агента**. Важно различать два последних понятия. *Функция агента* представляет собой абстрактное математическое описание, а *программа агента* — это конкретная реализация, действующая в рамках архитектуры агента.

Для иллюстрации изложенных идей воспользуемся очень простым примером: рассмотрим показанный на рис. 2.2 мир, в котором работает пылесос. Этот мир настолько прост, что существует возможность описать все, что в нем происходит; кроме того, это — мир, созданный человеком, поэтому можно изобрести множество вариантов его организации. В данном конкретном мире имеются только два местонахождения: квадраты A и B. Пылесос, выполняющий роль агента, воспринимает, в каком квадрате он находится и есть ли мусор в этом квадрате. Агент может выбрать такие действия, как переход влево, вправо, всасывание мусора или бездействие. Одна из очень простых функций агента состоит в следующем: если в текущем квадрате имеется мусор, то всосать его, иначе перейти в другой квадрат. Частичная табуляция данной функции агента показана в табл. 2.1. Простая программа агента для этой функции агента приведена ниже в этой главе, в листинге 2.2.

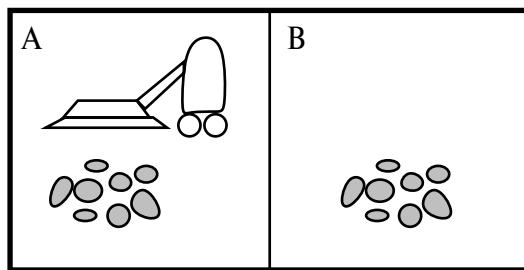


Рис. 2.2. Мир пылесоса, в котором имеются только два местонахождения

Таблица 2.1. Частичная табуляция функции простого агента для мира пылесоса, показанного на рис. 2.2

Последовательность актов восприятия	Действие
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
...	...
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
...	...

На основании табл. 2.1 можно сделать вывод, что для мира пылесоса можно определять различных агентов, заполняя разными способами правый столбец этой таблицы. Поэтому очевидный вопрос состоит в следующем: «Какой способ заполнения этой таблицы является правильным?» Иными словами, благодаря чему агент

становится хорошим или плохим, интеллектуальным или не соответствующим критериям интеллектуальности? Ответ на этот вопрос приведен в следующем разделе.

Прежде чем завершить этот раздел, необходимо отметить, что понятие агента рассматривается как инструмент для анализа систем, а не как абсолютная классификация, согласно которой мир делится на агентов и неагентов. Например, в качестве агента можно было бы рассматривать карманный калькулятор, который выбирает действие по отображению “4” после получения последовательности актов восприятия “ $2+2=$ ”, но подобный анализ вряд ли поможет понять работу калькулятора.

## 2.2. КАЧЕСТВЕННОЕ ПОВЕДЕНИЕ: КОНЦЕПЦИЯ РАЦИОНАЛЬНОСТИ

---

❖ **Рациональным агентом** является такой агент, который выполняет правильные действия; выражаясь более формально, таковым является агент, в котором каждая запись в таблице для функции агента заполнена правильно. Очевидно, что выполнение правильных действий лучше, чем осуществление неправильных действий, но что подразумевается под выражением “выполнение правильных действий”? В первом приближении можно сказать, что правильным действием является такое действие, которое обеспечивает наиболее успешное функционирование агента. Поэтому требуется определенный способ измерения успеха. Критерии успеха, наряду с описанием среды, а также датчиков и исполнительных механизмов агента, предоставляют полную спецификацию задачи, с которой сталкивается агент. Имея эти компоненты, мы можем определить более точно, что подразумевается под словом “рationalный”.

### Показатели производительности

❖ **Показатели производительности** воплощают в себе критерии оценки успешного поведения агента. После погружения в среду агент вырабатывает последовательность действий, соответствующих полученным им восприятиям. Эта последовательность действий вынуждает среду пройти через последовательность состояний. Если такая последовательность соответствует желаемому, то агент функционирует хорошо. Безусловно, что не может быть одного постоянного показателя, подходящего для всех агентов. Можно было бы узнать у агента его субъективное мнение о том, насколько он удовлетворен своей собственной производительностью, но некоторые агенты не будут способны ответить, а другие склонны заниматься самообманом<sup>2</sup>. Поэтому необходимо упорно добиваться применения объективных показателей производительности, и, как правило, проектировщик, конструирующий агента, предусматривает такие показатели.

Рассмотрим агент-пылесос, описанный в предыдущем разделе. Можно было бы предложить измерять показатели производительности по объему мусора, убранного за одну восьмичасовую смену. Но, безусловно, имея дело с рациональным агентом,

---

<sup>2</sup> Особенno известны тем, что недостигнутый успех для них — “зелен виноград”, такие агенты, как люди. Не получив кое-что для себя весьма ценное, они искренне считают, что и не стремились к этому: “Подумаешь! Мне и даром не нужна эта дурацкая Нобелевская премия!”

вы получаете то, что просите. Рациональный агент может максимизировать такой показатель производительности, убирая мусор, затем вываливая весь его на пол, затем снова убирая, и т.д. Поэтому более приемлемые критерии производительности должны вознаграждать агента за то, что пол остается чистым. Например, одно очко могло бы присуждаться за каждый чистый квадрат в каждом интервале времени (возможно, в сочетании со штрафом за потребляемую электроэнергию и создаваемый шум). *В качестве общего правила следует указать, что лучше всего разрабатывать показатели производительности в соответствии с тем, чего действительно необходимо добиться в данной среде, а не в соответствии с тем, как, по мнению проектировщика, должен вести себя агент.*

Задача выбора показателей производительности не всегда является простой. Например, понятие “чистого пола”, которое рассматривалось выше, основано на определении усредненной чистоты пола во времени. Но необходимо также учитывать, что одна и та же усредненная чистота может быть достигнута двумя различными агентами, один из которых постоянно, но неторопливо выполняет свою работу, а другой время от времени энергично занимается очисткой, но делает длинные перерывы. Может показаться, что определение того способа действий, который является в данном случае наиболее предпочтительным, относится к тонкостям домоводства, но фактически это — глубокий философский вопрос с далеко идущими последствиями. Что лучше — бесшабашная жизнь со взлетами и падениями или безопасное, но однообразное существование? Что лучше — экономика, в которой каждый живет в умеренной бедности, или такая экономика, в которой одни ни в чем не нуждаются, а другие еле сводят концы с концами? Оставляем задачу поиска ответов на эти вопросы в качестве упражнения для любознательного читателя.

## Рациональность

В любой конкретный момент времени оценка рациональности действий агента зависит от четырех перечисленных ниже факторов.

- Показатели производительности, которые определяют критерии успеха.
- Знания агента о среде, приобретенные ранее.
- Действия, которые могут быть выполнены агентом.
- Последовательность актов восприятия агента, которые произошли до настоящего времени.

С учетом этих факторов можно сформулировать следующее *определение рационального агента*.

*Для каждой возможной последовательности актов восприятия рациональный агент должен выбрать действие, которое, как ожидается, максимизирует его показатели производительности, с учетом фактов, предоставленных данной последовательностью актов восприятия и всех встроенных знаний, которыми обладает агент.*

Рассмотрим пример простого агента-пылесоса, который очищает квадрат, если в нем имеется мусор, и переходит в другой квадрат, если мусора в нем нет; результаты частичной табуляции такой функции агента приведены в табл. 2.1. Является ли этот агент рациональным? Ответ на этот вопрос не так уж прост! Вначале необходимо определить, в чем состоят показатели производительности, что известно о среде и ка-

кие датчики и исполнительные механизмы имеет агент. Примем перечисленные ниже предположения.

- Применяемые показатели производительности предусматривают вознаграждение в одно очко за каждый чистый квадрат в каждом интервале времени в течение “срока существования” агента, состоящего из 1000 интервалов времени.
- “География” среды известна заранее (рис. 2.2), но распределение мусора и первоначальное местонахождение агента не определены. Чистые квадраты остаются чистыми, а всасывание мусора приводит к очистке текущего квадрата. Действия *Left* и *Right* приводят к перемещению агента соответственно влево и вправо, за исключением тех случаев, когда они могли бы вывести агента за пределы среды, и в этих случаях агент остается там, где он находится.
- Единственными доступными действиями являются *Left*, *Right*, *Suck* (всосать мусор) и *NoOp* (ничего не делать).
- Агент правильно определяет свое местонахождение и воспринимает показания датчика, позволяющие узнать, имеется ли мусор в этом местонахождении.

Авторы утверждают, что в этих обстоятельствах агент действительно является рациональным; его ожидаемая производительность, по меньшей мере, не ниже, чем у любых других агентов. В упр. 2.4 предложено доказать это утверждение.

Можно легко обнаружить, что в других обстоятельствах тот же самый агент может стать нерациональным. Например, после того как весь мусор будет очищен, агент станет совершать ненужные периодические перемещения вперед и назад; если показатели производительности предусматривают штраф в одно очко за каждое передвижение в том или ином направлении, то агент не сможет хорошо зарабатывать. В таком случае лучший агент должен был бы ничего не делать до тех пор, пока он уверен в том, что все квадраты остаются чистыми. Если же чистые квадраты могут снова стать грязными, то агент обязан время от времени проводить проверку и снова очищать их по мере необходимости. А если география среды неизвестна, то агенту может потребоваться исследовать ее, а не ограничиваться квадратами *A* и *B*. В упр. 2.4 предложено спроектировать агентов для подобных случаев.

## Всезнание, обучение и автономность

Необходимо тщательно проводить различие между рациональностью и ~~всезнанием~~ всезнанием. Всезнающий агент знает фактический результат своих действий и может действовать соответствующим образом; но всезнание в действительности невозможно. Рассмотрим следующий пример: некий господин однажды гуляет в Париже по Елисейским Полям и видит на другой стороне улицы старого приятеля. Вблизи нет никаких машин, а наш господин никуда не спешит, поэтому, будучи рациональным агентом, он начинает переходить через дорогу. Между тем на высоте 10 000 метров у пролетающего самолета отваливается дверь грузового отсека<sup>3</sup>, и прежде чем несчастный успевает достичь другой стороны улицы, расплющивает его в лепешку. Было ли нерациональным именно то, что этот господин решил перейти на другую сторону улицы? Весьма маловероятно, что в его некрологе написали бы: “Жертва идиотской попытки перейти улицу”.

<sup>3</sup> См. заметку Н. Гендерсона “New door latches urged for Boeing 747 jumbo jets” (На дверях аэробусов Boeing 747 необходимо срочно установить новые замки). — Washington Post, 24 августа 1989 года.

Этот пример показывает, что рациональность нельзя рассматривать как равнозначную совершенству. Рациональность — это максимизация ожидаемой производительности, а совершенство — максимизация фактической производительности. Отказываясь от стремления к совершенству, мы не только применяем к агентам справедливые критерии, но и учитываем реальность. Дело в том, что если от агента требуют, чтобы он выполнял действия, которые оказываются наилучшими после их совершения, то задача проектирования агента, отвечающего этой спецификации, становится невыполнимой (по крайней мере, до тех пор, пока мы не сможем повысить эффективность машин времени или хрустальных шаров, применяемых гадалками).

Поэтому наше определение рациональности не требует всезнания, ведь рациональный выбор зависит только от последовательности актов восприятия, сформированной к данному моменту. Необходимо также следить за тем, чтобы мы непреднамеренно не позволили бы агенту участвовать в действиях, которые, безусловно, не являются интеллектуальными. Например, если агент не оглядывается влево и вправо, прежде чем пересечь дорогу с интенсивным движением, то полученная им до сих пор последовательность актов восприятия не сможет подсказать, что к нему на большой скорости приближается огромный грузовик. Указывает ли наше определение рациональности, что теперь агент может перейти через дорогу? Отнюдь нет! Во-первых, агент не был бы рациональным, если бы попытался перейти на другую сторону, получив такую неинформативную последовательность актов восприятия: риск несчастного случая при подобной попытке перейти автомагистраль, не оглянувшись, слишком велик. Во-вторых, рациональный агент должен выбрать действие “оглянуться”, прежде чем ступить на дорогу, поскольку такой осмотр позволяет максимизировать ожидаемую производительность. Выполнение в целях модификации будущих восприятий определенных действий (иногда называемых **сбором информации**) составляет важную часть рациональности и подробно рассматривается в главе 16. Второй пример сбора информации выражается в том **исследовании ситуации**, которое должно быть предпринято агентом-пылесосом в среде, которая первоначально была для него неизвестной.

Наше определение требует, чтобы рациональный агент не только собирал информацию, но также **обучался** в максимально возможной степени на тех данных, которые он воспринимает. Начальная конфигурация агента может отражать некоторые предварительные знания о среде, но по мере приобретения агентом опыта эти знания могут модифицироваться и пополняться. Существуют крайние случаи, в которых среда полностью известна заранее. В подобных случаях агенту не требуется воспринимать информацию или обучаться; он просто сразу действует правильно. Безусловно, такие агенты являются весьма уязвимыми. Рассмотрим скромного навозного жука. Выкопав гнездо и отложив яйца, он скатывает шарик навоза, набрав его из ближайшей навозной кучи, чтобы заткнуть вход в гнездо. Если шарик навоза будет удален непосредственно перед тем, как жук его схватит, жук продолжает манипулировать им и изображает такую пантомиму, как будто он затыкает гнездо несуществующим шариком навоза, даже не замечая, что этот шарик отсутствует. В результате эволюции поведение этого жука было сформировано на основании определенного предположения, а если это предположение нарушается, то за этим следует безуспешное поведение. Немного более интеллектуальными являются осы-сфексы. Самка сфекса выкапывает норку, выходит из нее, жалит гусеницу и затачивает ее в норку, затем снова выходит из норки, чтобы проверить, все ли в порядке, вытаски-

вает гусеницу наружу и откладывает в нее яйца. Гусеница служит в качестве источника питания во время развития яиц. До сих пор все идет хорошо, но если энтомолог переместит гусеницу на несколько дюймов в сторону, пока сфекс выполняет свою проверку, это насекомое снова возвращается к этапу “перетаскивания” своего плана и продолжает выполнять план без изменений, даже после десятков вмешательств в процедуру перемещения гусеницы. Оса-сфекс не способна обучиться действовать в такой ситуации, когда ее врожденный план нарушается, и поэтому не может его изменить.

В успешно действующих агентах задача вычисления функции агента разбивается на три отдельных периода: при проектировании агента некоторые вычисления осуществляются его проектировщиками; дополнительные вычисления агент производит, выбирая одно из своих очередных действий; а по мере того как агент учится на основании опыта, он осуществляет другие вспомогательные вычисления для принятия решения о том, как модифицировать свое поведение.

Если степень, в которой агент полагается на априорные знания своего проектировщика, а не на свои восприятия, слишком высока, то такой агент рассматривается как обладающий недостаточной **автономностью**. Рациональный агент должен быть автономным — он должен обучаться всему, что может освоить, для компенсации неполных или неправильных априорных знаний. Например, агент-пылесос, который обучается прогнозированию того, где и когда появится дополнительный мусор, безусловно, будет работать лучше, чем тот агент, который на это не способен. С точки зрения практики агенту редко предъявляется требование, чтобы он был полностью автономным с самого начала: если агент имеет мало опыта или вообще не имеет опыта, то вынужден действовать случайным образом, если проектировщик не оказал ему определенную помощь. Поэтому, как и эволюция предоставила животным достаточноное количество врожденных рефлексов, позволяющих им прожить после рождения настолько долго, чтобы успеть обучиться самостоятельно, так и профессиональному интеллектуальному агенту было бы разумно предоставить некоторые начальные знания, а не только наделить его способностью обучаться. После достаточного опыта существования в своей среде поведение рационального агента может по сути стать независимым от его априорных знаний. Поэтому включение в проект способности к обучению позволяет проектировать простых рациональных агентов, которые могут действовать успешно в исключительно разнообразных вариантах среды.

### **2.3. ОПРЕДЕЛЕНИЕ ХАРАКТЕРА СРЕДЫ**

---

Теперь, после разработки определения рациональности, мы почти готовы приступить к созданию рациональных агентов. Но вначале необходимо определить, чем является **проблемная среда**, по сути представляющая собой “проблему”, для которой рациональный агент служит “решением”. Начнем с демонстрации того, как определить проблемную среду, и проиллюстрируем этот процесс на ряде примеров. Затем в этом разделе будет показано, что проблемная среда может иметь целый ряд разновидностей. Выбор проекта, наиболее подходящего для программы конкретного агента, непосредственно зависит от рассматриваемой разновидности проблемной среды.

## Определение проблемной среды

В приведенном выше исследовании рациональности простого агента-пылесоса нам пришлось определить показатели производительности, среду, а также исполнительные механизмы и датчики агента. Сгруппируем описание всех этих факторов под заголовком **проблемная среда**. Для тех, кто любит аббревиатуры, авторы сокращенно обозначили соответствующее описание как **PEAS** (Performance, Environment, Actuators, Sensors — производительность, среда, исполнительные механизмы, датчики). Первый этап проектирования любого агента всегда должен состоять в определении проблемной среды с наибольшей возможной полнотой.

Пример, в котором рассматривался мир пылесоса, был несложным; теперь рассмотрим более сложную проблему — создание автоматизированного водителя такси. Этот пример будет использоваться во всей оставшейся части данной главы. Прежде чем читатель почувствует тревогу за безопасность будущих пассажиров, хотим сразу же отметить, что задача создания полностью автоматизированного водителя такси в настоящее время все еще выходит за пределы возможностей существующей технологии. (См. с. 69, где приведено описание существующего роботаводителя; с состоянием дел в этой области можно также ознакомиться по трудам конференций, посвященных интеллектуальным транспортным системам, в названиях которых есть слова *Intelligent Transportation Systems*.) Полное решение проблемы вождения автомобиля является чрезвычайно трудоемким, поскольку нет предела появления все новых и новых комбинаций обстоятельств, которые могут возникать в процессе вождения; это еще одна из причин, по которой мы выбрали данную проблему для обсуждения. В табл. 2.2 приведено итоговое описание PEAS для проблемной среды вождения такси. Каждый из элементов этого описания рассматривается более подробно в настоящей главе.

**Таблица 2.2. Описание PEAS проблемной среды для автоматизированного водителя такси**

Тип агента	Показатели производительности	Среда	Исполнительные механизмы	Датчики
Водитель такси	Безопасная, быстрая, комфортная езда в рамках правил дорожного движения, максимизация прибыли	Дороги, другие средства, пешеходы, клиенты	Рулевое управление, акселератор, тормоз, световые сигналы, звуковые сигналы, клаксон, дисплей	Видеокамеры, ультразвуковой дальномер, спидометр, глобальная система навигации и определения положения, одометр, акселерометр, датчики двигателя, клавиатура

Прежде всего необходимо определить **показатели производительности**, которыми мы могли бы стимулировать деятельность нашего автоматизированного водителя. К желаемым качествам относится успешное достижение нужного места назначения; минимизация потребления топлива, износа и старения; минимизация продолжительности и/или стоимости поездки; минимизация количества нарушений правил дорожного движения и помех другим водителям; максимизация безопасности и комфорта пассажиров; максимизация прибыли. Безусловно, что некоторые из этих целей конфликтуют, поэтому должны рассматриваться возможные компромиссы.

Затем рассмотрим, в чем состоит **среда вождения**, в которой действует такси. Любому водителю такси приходится иметь дело с самыми различными дорогами, начи-

ная с проселков и узких городских переулков и заканчивая автострадами с двенадцатью полосами движения. На дороге встречаются другие транспортные средства, беспризорные животные, пешеходы, рабочие, производящие дорожные работы, полицейские автомобили, лужи и выбоины. Водителю такси приходится также иметь дело с потенциальными и действительными пассажирами. Кроме того, имеется еще несколько важных дополнительных факторов. Таксисту может выпасть участь работать в Южной Калифорнии, где редко возникает такая проблема, как снег, или на Аляске, где снега на дорогах не бывает очень редко. Может оказаться, что водителю всю жизнь придется ездить по правой стороне или от него может потребоваться, чтобы он сумел достаточно успешно приспособиться к езде по левой стороне во время пребывания в Британии или в Японии. Безусловно, чем более ограниченной является среда, тем проще задача проектирования.

**Исполнительные механизмы**, имеющиеся в автоматизированном такси, должны быть в большей или меньшей степени такими же, как и те, которые находятся в распоряжении водителя-человека: средства управления двигателем с помощью акселератора и средства управления вождением с помощью руля и тормозов. Кроме того, для него могут потребоваться средства вывода на экран дисплея или синтеза речи для передачи ответных сообщений пассажирам и, возможно, определенные способы общения с водителями других транспортных средств, иногда вежливого, а иногда и не совсем.

Для достижения своих целей в данной среде вождения таксисту необходимо будет знать, где он находится, кто еще едет по этой дороге и с какой скоростью движется он сам. Поэтому в число его основных **датчиков** должны входить одна или несколько управляемых телевизионных камер, спидометр и одометр. Для правильного управления автомобилем, особенно на поворотах, в нем должен быть предусмотрен акселерометр; водителю потребуется также знать механическое состояние автомобиля, поэтому для него будет нужен обычный набор датчиков для двигателя и электрической системы. Автоматизированный водитель может также иметь приборы, недоступные для среднего водителя-человека: спутниковую глобальную систему навигации и определения положения (Global Positioning System — GPS) для получения точной информации о местонахождении по отношению к электронной карте, а также инфракрасные или ультразвуковые датчики для измерения расстояний до других автомобилей и препятствий. Наконец, ему потребуется клавиатура или микрофон для пассажиров, чтобы они могли указать место своего назначения.

В табл. 2.3 кратко перечислены основные элементы PEAS для целого ряда других типов агентов. Дополнительные примеры приведены в упр. 2.5. Некоторым читателям может показаться удивительным то, что авторы включили в этот список типов агентов некоторые программы, которые функционируют в полностью искусственной среде, ограничивающей вводом с клавиатуры и выводом символов на экран. Кое-кто мог бы сказать: “Разумеется, это же не реальная среда, не правда ли?” В действительности суть состоит не в различиях между “реальными” и “искусственными” вариантами среды, а в том, какова сложность связей между поведением агента, последовательностью актов восприятия, вырабатываемой этой средой, и показателями производительности. Некоторые “реальные” варианты среды фактически являются чрезвычайно простыми. Например, для робота, предназначенного для контроля деталей, проходящих мимо него на ленточном конвейере, может использоваться целый ряд упрощающих допущений, например, что освещение всегда включено, что единственны-

ми предметами на ленте конвейера являются детали того типа, который ему известен, и что существуют только два действия (принять изделие или забраковать его).

Таблица 2.3. Примеры типов агентов и их описаний PEAS

Тип агента	Показатели производительности	Среда	Исполнительные механизмы	Датчики
Медицинская диагностическая система	Успешное излечение пациента, минимизация затрат, отсутствие поводов для судебных тяжб	Пациент, больница, персонал	Вывод вопросов, тестов, диагнозов, рекомендаций, направлений	Ввод с клавиатуры симптомов, результатов лабораторных исследований, ответов пациента
Система анализа изображений, полученных со спутника	Правильная классификация изображения	Канал передачи данных от орбитального спутника	Вывод на дисплей результатов классификации определенного фрагмента изображения	Массивы пикселей с данными о цвете
Робот-сортировщик деталей	Процентные показатели безошибочной сортировки по лоткам	Ленточный конвейер с движущимися на нем деталями; лотки	Шарнирный манипулятор и захват	Видеокамера, датчики углов поворота шарниров
Контроллер очистительной установки	Максимизация степени очистки, продуктивности, безопасности	Очистительная установка, операторы	Клапаны, насосы, нагреватели, дисплеи	Температура, давление, датчики химического состава
Интерактивная программа обучения английскому языку	Максимизация оценок студентов на экзаменах	Множество студентов, экзаменационное агентство	Вывод на дисплей упражнений, рекомендаций, исправлений	Ввод с клавиатуры

В отличие от этого некоторые **программные агенты** (называемые также **программными роботами** или **софтботами**) существуют в сложных, неограниченных проблемных областях. Представьте себе программный робот, предназначенный для управления тренажером, имитирующим крупный пассажирский самолет. Этот тренажер представляет собой очень детально промоделированную, сложную среду, в которой имитируются движения других самолетов и работа наземных служб, а программный агент должен выбирать в реальном времени наиболее целесообразные действия из широкого диапазона действий. Еще одним примером может служить программный робот, предназначенный для просмотра источников новостей в Internet и показа клиентам интересующих их сообщений. Для успешной работы ему требуются определенные способности к обработке текста на естественном языке, он должен в процессе обучения определять, что интересует каждого заказчика, а также должен уметь изменять свои планы динамически, допустим, когда соединение с каким-либо из источников новостей закрывается или в оперативный режим переходит новый источник новостей. Internet представляет собой среду, которая по своей сложности соперничает с физическим миром, а в число обитателей этой сети входит много искусственных агентов.

## Свойства проблемной среды

Несомненно, что разнообразие вариантов проблемной среды, которые могут возникать в искусственном интеллекте, весьма велико. Тем не менее существует возможность определить относительно небольшое количество измерений, по которым могут быть классифицированы варианты проблемной среды. Эти измерения в значительной степени определяют наиболее приемлемый проект агента и применимость каждого из основных семейств методов для реализации агента. Вначале в этом разделе будет приведен список измерений, а затем проанализировано несколько вариантов проблемной среды для иллюстрации этих идей. Приведенные здесь определения являются неформальными; более точные утверждения и примеры вариантов среды каждого типа описаны в следующих главах.

- **☒ Полностью наблюдаемая или частично наблюдаемая**

Если датчики агента предоставляют ему доступ к полной информации о состоянии среды в каждый момент времени, то такая проблемная среда называется полностью наблюдаемой<sup>4</sup>. По сути, проблемная среда является полностью наблюдаемой, если датчики выявляют все данные, которые являются релевантными для выбора агентом действия; релевантность, в свою очередь, зависит от показателей производительности. Полностью наблюдаемые варианты среды являются удобными, поскольку агенту не требуется поддерживать какое-либо внутреннее состояние для того, чтобы быть в курсе всего происходящего в этом мире. Среда может оказаться частично наблюдаемой из-за создающих шум и неточных датчиков или из-за того, что отдельные характеристики ее состояния просто отсутствуют в информации, полученной от датчиков; например, агент-пылесос, в котором имеется только локальный датчик мусора, не может определить, имеется ли мусор в других квадратах, а автоматизированный водитель такси не имеет сведений о том, какие маневры намереваются выполнить другие водители.

- **☒ Детерминированная или ☒ стохастическая**

Если следующее состояние среды полностью определяется текущим состоянием и действием, выполненным агентом, то такая среда называется детерминированной; в противном случае она является стохастической. В принципе в полностью наблюдаемой детерминированной среде агенту не приходится действовать в условиях неопределенности. Но если среда — частично наблюдаемая, то может создаться впечатление, что она является стохастической. Это отчасти справедливо, если среда — сложная и агенту нелегко следить за всеми ее ненаблюдаемыми аспектами. В связи с этим часто бывает более удобно классифицировать среду как детерминированную или стохастическую с точки зрения агента. Очевидно, что при такой трактовке среда вождения такси является стохастической, поскольку никто не может точно предсказать поведение всех других транспортных средств; более того, в любом автомобиле совершенно неожиданно может произойти прокол шины или остановка двигателя.

<sup>4</sup> В первом издании этой книги использовались термины **доступная** среда и **недоступная** среда вместо терминов **полностью наблюдаемая** среда и **частично наблюдаемая** среда; **недетерминированная** вместо **стохастической** и **неэпизодическая** вместо **последовательной**. Применяемая в этом издании новая терминология более совместима со сложившейся в этой области терминологией.

Описанный здесь мир пылесоса является детерминированным, но другие варианты этой среды могут включать стохастические элементы, такие как случайное появление мусора и ненадежная работа механизма всасывания (см. упр. 2.12). Если среда является детерминированной во всех отношениях, кроме действий других агентов, то авторы данной книги называют эту среду **стратегической**.

- **Эпизодическая или последовательная<sup>5</sup>**

В эпизодической проблемной среде опыт агента состоит из неразрывных эпизодов. Каждый эпизод включает в себя восприятие среды агентом, а затем выполнение одного действия. При этом крайне важно то, что следующий эпизод не зависит от действий, предпринятых в предыдущих эпизодах. В эпизодических вариантах среды выбор действия в каждом эпизоде зависит только от самого эпизода. Эпизодическими являются многие задачи классификации. Например, агент, который должен распознавать дефектные детали на сборочной линии, формирует каждое решение применительно к текущей детали, независимо от предыдущих решений; более того, от текущего решения не зависит то, будет ли определена как дефектная следующая деталь. С другой стороны, в последовательных вариантах среды текущее решение может повлиять на все будущие решения. Последовательными являются такие задачи, как игра в шахматы и вождение такси: в обоих случаях кратковременные действия могут иметь долговременные последствия. Эпизодические варианты среды гораздо проще по сравнению с последовательными, поскольку в них агенту не нужно думать наперед.

- **Статическая или динамическая**

Если среда может измениться в ходе того, как агент выбирает очередное действие, то такая среда называется динамической для данного агента; в противном случае она является статической. Действовать в условиях статической среды проще, поскольку агенту не требуется наблюдать за миром в процессе выработки решения о выполнении очередного действия, к тому же агенту не приходится беспокоиться о том, что он затрачивает на размышления слишком много времени. Динамические варианты среды, с другой стороны, как бы непрерывно спрашивают агента, что он собирается делать, а если он еще ничего не решил, то это рассматривается как решение ничего не делать. Если с течением времени сама среда не изменяется, а изменяются показатели производительности агента, то такая среда называется **полудинамической**. Очевидно, что среда вождения такси является динамической, поскольку другие автомобили и само такси продолжают движение и в ходе того, как алгоритм вождения определяет, что делать дальше. Игра в шахматы с контролем времени является полудинамической, а задача решения кроссворда — статической.

- **Дискретная или непрерывная**

Различие между дискретными и непрерывными вариантами среды может относиться к состоянию среды, способу учета времени, а также восприятиям и действиям агента. Например, такая среда с дискретными состояниями, как

---

<sup>5</sup> Термин “последовательный” используется также в компьютерных науках как антоним термина “параллельный”. Соответствующие два толкования фактически не связаны друг с другом.

игра в шахматы, имеет конечное количество различных состояний. Кроме того, игра в шахматы связана с дискретным множеством восприятий и действий. Вождение такси — это проблема с непрерывно меняющимся состоянием и непрерывно текущим временем, поскольку скорость и местонахождение самого такси и других транспортных средств изменяются в определенном диапазоне непрерывных значений, причем эти изменения происходят во времени плавно. Действия по вождению такси также являются непрерывными (непрерывная регулировка угла поворота руля и т.д.). Строго говоря, входные данные от цифровых камер поступают дискретно, но обычно рассматриваются как представляющие непрерывно изменяющиеся скорости и местонахождения.

- **❖ Одноагентная или ❖ мультиагентная**

Различие между одноагентными и мультиагентными вариантами среды на первый взгляд может показаться достаточно простым. Например, очевидно, что агент, самостоятельно решающий кроссворд, находится в одноагентной среде, а агент, играющий в шахматы, действует в двухагентной среде. Тем не менее при анализе этого классификационного признака возникают некоторые нюансы. Прежде всего, выше было описано, на каком основании некоторая сущность *может* рассматриваться как агент, но не было указано, какие сущности *должны* рассматриваться как агенты. Должен ли агент A (например, водитель такси) считать агентом объект B (другой автомобиль), или может относиться к нему просто как к стохастически действующему объекту, который можно сравнить с волнами, набегающими на берег, или с листьями, трепещущими на ветру? Ключевое различие состоит в том, следует ли или не следует описывать поведение объекта B как максимизирующее личные показатели производительности, значения которых зависят от поведения агента A. Например, в шахматах соперничающая сущность B пытается максимизировать свои показатели производительности, а это по правилам шахмат приводит к минимизации показателей производительности агента A. Таким образом, шахматы — это **❖ конкурентная** мультиагентная среда. А в среде вождения такси, с другой стороны, предотвращение столкновений максимизирует показатели производительности всех агентов, поэтому она может служить примером частично **❖ кооперативной** мультиагентной среды. Она является также частично конкурентной, поскольку, например, парковочную площадку может занять только один автомобиль. Проблемы проектирования агентов, возникающие в мультиагентной среде, часто полностью отличаются от тех, с которыми приходится сталкиваться в одноагентных вариантах среды; например, одним из признаков рационального поведения в мультиагентной среде часто бывает **поддержка связи**, а в некоторых вариантах частично наблюдавшей конкурентной среды рациональным становится **стохастическое поведение**, поскольку оно позволяет избежать ловушек *предсказуемости*.

Как и следует ожидать, наиболее сложными вариантами среды являются частично наблюдавшиеся, стохастические, последовательные, динамические, непрерывные и мультиагентные. Кроме того, часто обнаруживается, что многие реальные ситуации являются настолько сложными, что неясно даже, действительно ли их можно считать детерминированными. С точки зрения практики их следует рассматривать как стохастические. Проблема вождения такси является сложной во всех указанных отношениях.

В табл. 2.4 перечислены свойства многих известных вариантов среды. Следует отметить, что в отдельных случаях приведенные в ней описания являются слишком краткими и сухими. Например, в ней указано, что шахматы — это полностью наблюдаемая среда, но строго говоря, это утверждение является ложным, поскольку некоторые правила, касающиеся рокировки, взятия пешки на проходе и объявления ничьи при повторении ходов, требуют запоминания определенных фактов об истории игры, которые нельзя выявить из анализа позиции на доске. Но эти исключения из определения наблюдаемости, безусловно, являются незначительными по сравнению с теми необычными ситуациями, с которыми сталкивается автоматизированный водитель такси, интерактивная система преподавания английского языка или медицинская диагностическая система.

Таблица 2.4. Примеры вариантов проблемной среды и их характеристик

Проблемная среда	Наблюдаемая полностью или частично	Детерминированная, стратегическая или стохастическая	Эпизодическая или последовательная	Статическая, динамическая или полудинамическая	Дискретная или непрерывная	Одноагентная или мультиагентная
Решение кроссворда	Полностью наблюдаемая	Детерминированная	Последовательная	Статическая	Дискретная	Одноагентная
Игра в шахматы с контролем времени	Полностью наблюдаемая	Стохастическая	Последовательная	Полудинамическая	Дискретная	Мультиагентная
Игра в покер	Частично наблюдаемая	Стохастическая	Последовательная	Статическая	Дискретная	Мультиагентная
Игра в нарды	Полностью наблюдаемая	Стохастическая	Последовательная	Статическая	Дискретная	Мультиагентная
Вождение такси	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Непрерывная	Мультиагентная
Медицинская диагностика	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Непрерывная	Одноагентная
Анализ изображений	Полностью наблюдаемая	Детерминированная	Эпизодическая	Полудинамическая	Непрерывная	Одноагентная
Робот-сортировщик деталей	Частично наблюдаемая	Стохастическая	Эпизодическая	Динамическая	Непрерывная	Одноагентная
Контроллер очистительной установки	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Непрерывная	Одноагентная
Интерактивная программа, обучающая английскому языку	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Дискретная	Мультиагентная

Некоторые другие ответы в этой таблице зависят от того, как определена проблемная среда. Например, в ней задача медицинского диагноза определена как одногентная, поскольку сам процесс развития заболевания у пациента нецелесообразно моделировать в качестве агента, но системе медицинской диагностики иногда приходится сталкиваться с пациентами, не желающими принимать ее рекомендации, и со скептически настроенным персоналом, поэтому ее среда может иметь мультиагентный аспект. Кроме того, медицинская диагностика является эпизодической, если она рассматривается как задача выбора диагноза на основе анализа перечня симптомов, но эта проблема становится последовательной, если решаемая при этом задача может включать выработку рекомендаций по выполнению ряда лабораторных исследований, оценку прогресса в ходе лечения и т.д. К тому же многие варианты среды являются эпизодическими на более высоких уровнях по сравнению с отдельными действиями агента. Например, шахматный турнир состоит из ряда игр; каждая игра является эпизодом, поскольку (вообще говоря) от ходов, сделанных в предыдущей игре, не зависит то, как повлияют на общую производительность агента ходы, сделанные им в текущей игре. С другой стороны, принятие решений в одной и той же игре, безусловно, происходит последовательно.

Репозитарий кода, который относится к данной книге ([aima.cs.berkeley.edu](http://aima.cs.berkeley.edu)), включает реализации многих вариантов среды, наряду с имитатором среды общего назначения, который помещает одного или нескольких агентов в моделируемую среду, наблюдает за их поведением в течение определенного времени и оценивает их действия в соответствии с заданными показателями производительности. Такие эксперименты часто выполняются применительно не к одному варианту среды, а ко многим вариантам, сформированным на основе некоторого **класса вариантов среды**. Например, чтобы оценить действия водителя такси в моделируемой ситуации дорожного движения, может потребоваться провести много сеансов моделирования с различными условиями трафика, освещения и погоды. Если бы мы спроектировали этого агента для одного сценария, то могли бы лучше воспользоваться специфическими свойствами данного конкретного случая, но не имели бы возможности определить приемлемый проект решения задачи автоматизированного вождения в целом. По этой причине репозитарий кода включает также **генератор вариантов среды** для каждого класса вариантов среды; этот генератор выбирает определенные варианты среды (с некоторой вероятностью), в которых выполняется проверка агента. Например, генератор вариантов среды пылесоса инициализирует случайным образом такие исходные данные, как распределение мусора и местонахождение агента. Дело в том, что наибольший интерес представляет то, какую среднюю производительность будет иметь данный конкретный агент в некотором классе вариантов среды. Рациональный агент для определенного класса вариантов среды максимизирует свою среднюю производительность. Процесс разработки класса вариантов среды и оценки в них различных агентов иллюстрируется в упр. 2.7–2.12.

---

## 2.4. СТРУКТУРА АГЕНТОВ

---

До сих пор в этой книге свойства агентов рассматривались на основании анализа их поведения — действий, выполняемых агентом после получения любой заданной последовательности актов восприятия. Теперь нам поневоле придется сменить тему

и перейти к описанию того, как организовано их внутреннее функционирование. Задача искусственного интеллекта состоит в разработке **программы агента**, которая реализует функцию агента, отображая восприятия на действия. Предполагается, что эта программа должна работать в своего рода вычислительном устройстве с физическими датчиками и исполнительными механизмами; в целом эти компоненты именуются в данной книге **архитектурой**, а структура агента условно обозначается следующей формулой:

$$\text{Агент} = \text{Архитектура} + \text{Программа}$$

Очевидно, что выбранная программа должна быть подходящей для этой архитектуры. Например, если в программе осуществляется выработка рекомендаций по выполнению таких действий, как *Walk* (ходьба), то в архитектуре целесообразно предусмотреть использование опорно-двигательного аппарата. Архитектура может представлять собой обычный персональный компьютер или может быть воплощена в виде роботизированного автомобиля с несколькими бортовыми компьютерами, видеокамерами и другими датчиками. Вообще говоря, архитектура обеспечивает передачу в программу результатов восприятия, полученных от датчиков, выполнение программы и передачу исполнительным механизмам вариантов действий, выбранных программой, по мере их выработки. Основная часть данной книги посвящена проектированию программ агентов, а главы 24 и 25 касаются непосредственно датчиков и исполнительных механизмов.

## Программы агентов

Все программы агентов, которые будут разработаны в этой книге, имеют одну и ту же структуру: они принимают от датчиков в качестве входных данных результаты текущего восприятия и возвращают исполнительным механизмам выбранный вариант действия<sup>6</sup>. Необходимо указать на различие между программой агента, которая принимает в качестве входных данных результаты текущего восприятия, и функцией агента, которая принимает на входе всю историю актов восприятия. Программа агента получает в качестве входных данных только результаты текущего восприятия, поскольку больше ничего не может узнать из своей среды; если действия агента зависят от всей последовательности актов восприятия, то агент должен сам запоминать результаты этих актов восприятия.

Для описания программ агентов будет применяться простой язык псевдокода, который определен в приложении Б. (Оперативный репозитарий кода содержит реализации на реальных языках программирования.) Например, в листинге 2.1 показана довольно несложная программа агента, которая регистрирует последовательность актов восприятия, а затем использует полученную последовательность для доступа по индексу к таблице действий и определения того, что нужно сделать. Таблица явно отображает функцию агента, воплощаемую данной программой агента. Чтобы создать рационального агента таким образом, проектировщики должны сформировать

<sup>6</sup> Существуют и другие варианты структуры программы агента, например, можно было бы использовать в виде программ агентов **сопроцедуры**, которые действуют асинхронно со средой. Каждая такая сопроцедура имеет входной и выходной порты, а ее работа организована в виде цикла, в котором из входного портачитываются результаты восприятий, а в выходной порт записываются варианты действий.

таблицу, которая содержит подходящее действие для любой возможной последовательности актов восприятия.

**Листинг 2.1. Программа Table-Driven-Agent, которая вызывается после каждого восприятия новых данных и каждый раз возвращает вариант действия; программа регистрирует последовательность актов восприятия с использованием своей собственной закрытой структуры данных**

---

```

function Table-Driven-Agent(percept) returns действие action
    static: percepts, последовательность актов восприятия,
             первоначально пустая
    table, таблица действий, индексированная по
             последовательностям актов восприятия и
             полностью заданная с самого начала

    добавить результаты восприятия percept к концу
    последовательности percepts
    action  $\leftarrow$  Lookup(percepts, table)
    return action

```

---

Анализ того, почему такой подход к созданию агента, основанный на использовании таблицы, обречен на неудачу, является весьма поучительным. Допустим, что  $\mathcal{P}$  — множество возможных актов восприятия, а  $T$  — срок существования агента (общее количество актов восприятия, которое может быть им получено). Поисковая таблица будет содержать

$$\sum_{t=1}^T |\mathcal{P}|^t \text{ записей.}$$

Рассмотрим автоматизированное такси: визуальные входные данные от одной камеры поступают со скоростью примерно 27 мегабайтов в секунду (30 кадров в секунду,  $640 \times 480$  пикселов с 24 битами информации о цвете). Согласно этим данным поисковая таблица, рассчитанная на 1 час вождения, должна содержать количество записей, превышающее  $10^{250\ 000\ 000\ 000}$ . И даже поисковая таблица для шахмат (крошечного, хорошо изученного фрагмента реального мира) имела бы, по меньшей мере,  $10^{150}$  записей. Ошеломляющий размер этих таблиц (притом что количество атомов в наблюдаемой вселенной не превышает  $10^{80}$ ) означает, что, во-первых, ни один физический агент в нашей вселенной не имеет пространства для хранения такой таблицы, во-вторых, проектировщик не сможет найти достаточно времени для создания этой таблицы, в-третьих, ни один агент никогда не сможет обучиться тому, что содержится во всех правильных записях этой таблицы, на основании собственного опыта, и, в-четвертых, даже если среда достаточно проста для того, чтобы можно было создать таблицу приемлемых размеров, все равно у проектировщика нет руководящих сведений о том, как следует заполнять записи подобной таблицы.

Несмотря на все сказанное, программа Table-Driven-Agent выполняет именно то, что от нее требуется: она реализует желаемую функцию агента. Основная сложность, стоящая перед искусственным интеллектом как научным направлением, состоит в том, чтобы узнать, как создавать программы, которые в рамках возможного вырабатывают рациональное поведение с использованием небольшого объема кода, а не большого количества записей таблицы. Существует множество примеров, показывающих, что такая задача может быть выполнена успешно в других областях; например, огромные таблицы квадратных корней, использовавшиеся инженерами и

школьниками до 1970-х годов, теперь заменены работающей в электронных калькуляторах программой из пяти строк, в которой применяется метод Ньютона. Вопрос заключается в том, может ли искусственный интеллект сделать для интеллектуального поведения в целом то, что Ньютон сделал для упрощения вычисления квадратных корней? Авторы данной книги полагают, что ответ на этот вопрос является положительным.

В остальной части этого раздела рассматриваются четыре основных вида программ агентов, которые воплощают принципы, лежащие в основе почти всех интеллектуальных систем:

- простые рефлексные агенты;
- рефлексные агенты, основанные на модели;
- агенты, действующие на основе цели;
- агенты, действующие на основе полезности.

Затем приведено описание в общих терминах того, как преобразовать агентов всех этих типов в обучающихся агентов.

## Простые рефлексные агенты

Простейшим видом агента является **простой рефлексный агент**. Подобные агенты выбирают действия на основе текущего акта восприятия, игнорируя всю остальную историю актов восприятия. Например, агент-пылесос, для которого результаты табуляции функции агента приведены в табл. 2.1, представляет собой простой рефлексный агент, поскольку его решения основаны только на информации о текущем местонахождении и о том, содержит ли оно мусор. Программа для данного агента приведена в листинге 2.2.

**Листинг 2.2. Программа простого рефлексного агента в среде пылесоса с двумя состояниями. Эта программа реализует функцию агента, которая табулирована в табл. 2.1**

---

```
function Reflex-Vacuum-Agent([location,status]) returns действие
action

    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

---

Обратите внимание на то, что эта программа агента-пылесоса действительно очень мала по сравнению с соответствующей таблицей. Наиболее очевидное сокращение обусловлено тем, что в ней игнорируется история актов восприятия, в результате чего количество возможных вариантов сокращается от  $4^T$  просто до 4. Дополнительное небольшое сокращение обусловлено тем фактом, что если в текущем квадрате имеется мусор, то выполняемое при этом действие не зависит от местонахождения пылесоса.

Представьте себя на месте водителя автоматизированного такси. Если движущийся впереди автомобиль тормозит и загораются его тормозные огни, то вы должны заметить это и тоже начать торможение. Иными словами, над визуальными входными данными выполняется определенная обработка для выявления условия,

которое обозначается как “*car-in-front-is-braking*” (движущийся впереди автомобиль тормозит). Затем это условие активизирует некоторую связь с действием “*initiate-braking*” (начать торможение), установленную в программе агента. Такая связь называется **правилом условие–действие**<sup>7</sup> и записывается следующим образом:

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

Люди также используют большое количество таких связей, причем некоторые из них представляют собой сложные отклики, освоенные в результате обучения (как при рождении автомобиля), а другие являются врожденными рефлексами (такими как моргание, которое происходит при приближении к глазу постороннего предмета). В разных главах данной книги будет показано несколько различных способов, с помощью которых можно организовать обучение агента и реализацию таких связей.

Программа, приведенная в листинге 2.2, специализирована для одной конкретной среды пылесоса. Более общий и гибкий подход состоит в том, чтобы вначале создать интерпретатор общего назначения для правил условие–действие, а затем определить наборы правил для конкретной проблемной среды. На рис. 2.3 приведена структура такой общей программы в схематической форме и показано, каким образом правила условие–действие позволяют агенту создать связь от восприятия к действию. (Не следует беспокоиться, если такой способ покажется тривиальным; вскоре он обнаружит намного более интересные возможности.) В подобных схемах для обозначения текущего внутреннего состояния процесса принятия решения агентом используются прямоугольники, а для представления фоновой информации, применяемой в этом процессе, служат овалы. Программа этого агента, которая также является очень простой, приведена в листинге 2.3. Функция *Interpret-Input* вырабатывает абстрагированное описание текущего состояния по результатам восприятия, а функция *Rule-Match* возвращает первое правило во множестве правил, которое соответствует заданному описанию состояния. Следует отметить, что приведенное здесь изложение в терминах “правил” и “соответствия” является чисто концептуальным; фактические реализации могут быть настолько простыми, как совокупность логических элементов, реализующих логическую схему.

**Листинг 2.3. Программа простого рефлексного агента, который действует согласно правилу, условие которого соответствует текущему состоянию, определяемому результатом восприятия**

---

```
function Simple-Reflex-Agent(percept) returns действие action
  static: rules, множество правил условие–действие

  state  $\leftarrow$  Interpret-Input(percept)
  rule  $\leftarrow$  Rule-Match(state, rules)
  action  $\leftarrow$  Rule-Action[rule]
  return action
```

---

Простые рефлексные агенты характеризуются той замечательной особенностью, что они чрезвычайно просты, но зато обладают весьма ограниченным интеллектом. Агент, программа которого приведена в листинге 2.3, работает, ~~если~~ только если правильное решение может быть принято на основе исключительно текущего восприятия,

<sup>7</sup> Эти связи называются также **правилами ситуация–действие**, **продукциями** или **правилами if-then**.

иначе говоря, только если среда является полностью наблюдаемой. Внесение даже небольшой доли ненаблюдаемости может вызвать серьезное нарушение его работы. Например, в приведенном выше правиле торможения принято предположение, что условие *car-in-front-is-braking* может быть определено из текущего восприятия (текущего видеоизображения), если движущийся автомобиль имеет тормозной сигнал, расположенный на центральном месте среди других сигналов. К сожалению, некоторые более старые модели имеют другие конфигурации задних фар, тормозных сигналов, габаритных огней, сигналов торможения и сигналов поворота, поэтому не всегда возможно определить из единственного изображения, тормозит ли этот автомобиль или нет. Простой рефлексный агент, ведущий свой автомобиль вслед за таким автомобилем, либо будет постоянно тормозить без всякой необходимости, либо, что еще хуже, вообще не станет тормозить.

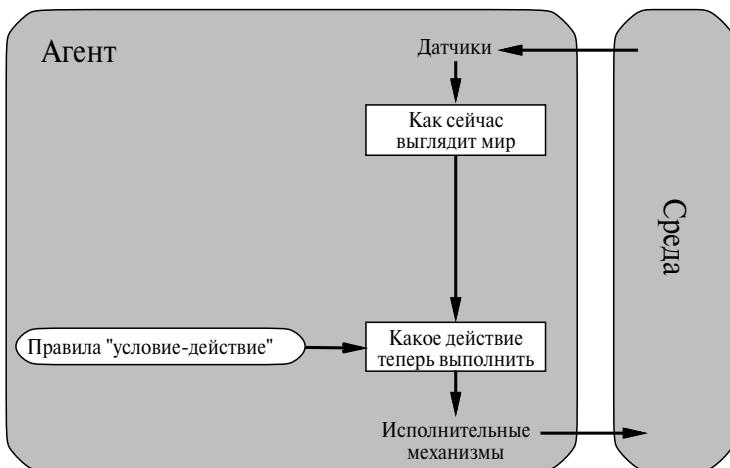


Рис. 2.3. Схематическое изображение структуры простого рефлексного агента

Возникновение аналогичной проблемы можно обнаружить и в мире пылесоса. Предположим, что в простом рефлексном агенте-пылесосе испортился датчик местонахождения и работает только датчик мусора. Такой агент получает только два возможных восприятия: [*Dirty*] и [*Clean*]. Он может выполнить действие *Suck* в ответ на восприятие [*Dirty*], а что он должен делать в ответ на восприятие [*Clean*]? Выполнение движения *Left* завершится отказом (на неопределенное долгое время), если окажется, что он начинает это движение с квадрата A, а если он начинает движение с квадрата B, то завершится отказом на неопределенное долгое время движение *Right*. Для простых рефлексных агентов, действующих в частично наблюдаемых вариантах среды, часто бывают неизбежными бесконечные циклы.

Выход из бесконечных циклов становится возможным, если агент обладает способностью **randomизировать** свои действия (вводить в них элемент случайности). Например, если агент-пылесос получает результат восприятия [*Clean*], то может подбросить монету, чтобы выбрать между движениями *Left* и *Right*. Легко показать, что агент достигнет другого квадрата в среднем за два этапа. Затем, если в этом квадрате имеется мусор, то пылесос его уберет и задача очистки будет выполнена.

Поэтому рандомизированный простой рефлексный агент может превзойти по своей производительности детерминированного простого рефлексного агента.

В разделе 2.3 уже упоминалось, что в некоторых мультиагентных вариантах среды может оказаться рациональным рандомизированное поведение правильного типа. А в одноагентных вариантах среды рандомизация обычно не является рациональной. Это лишь полезный трюк, который помогает простому рефлексному агенту в некоторых ситуациях, но в большинстве случаев можно добиться гораздо большего с помощью более сложных детерминированных агентов.

### Рефлексные агенты, основанные на модели

Наиболее эффективный способ организации работы в условиях частичной наблюдаемости состоит в том, чтобы агент отслеживал ту часть мира, которая воспринимается им в текущий момент. Это означает, что агент должен поддерживать своего рода **внутреннее состояние**, которое зависит от истории актов восприятия и поэтому отражает по крайней мере некоторые из ненаблюдаемых аспектов текущего состояния. Для решения задачи торможения поддержка внутреннего состояния не требует слишком больших затрат — для этого достаточно сохранить предыдущий кадр, снятый видеокамерой, чтобы агент мог определить тот момент, когда два красных световых сигнала с обеих сторон задней части идущего впереди автомобиля загораются или гаснут одновременно. Для решения других задач вождения, таких как переход с одной полосы движения на другую, агент должен следить за тем, где находятся другие автомобили, если он не может видеть все эти автомобили одновременно.

Для обеспечения возможности обновления этой внутренней информации о состоянии в течение времени необходимо, чтобы в программе агента были закодированы знания двух видов. Во-первых, нужна определенная информация о том, как мир изменяется независимо от агента, например, о том, что автомобиль, идущий на обгон, обычно становится ближе, чем в какой-то предыдущий момент. Во-вторых, требуется определенная информация о том, как влияют на мир собственные действия агента, например, что при повороте агентом рулевого колеса по часовой стрелке автомобиль поворачивает вправо или что после проезда по автомагистрали в течение пяти минут на север автомобиль находится на пять миль севернее от того места, где он был пять минут назад. Эти знания о том, “как работает мир” (которые могут быть воплощены в простых логических схемах или в сложных научных теориях) называются **моделью мира**. Агент, в котором используется такая модель, называется **агентом, основанным на модели**.

На рис. 2.4 приведена структура рефлексного агента, действующего с учетом внутреннего состояния, и показано, как текущее восприятие комбинируется с прежним внутренним состоянием для выработки обновленного описания текущего состояния. Программа такого агента приведена в листинге 2.4. В этом листинге интерес представляет функция `Update-State`, которая отвечает за создание нового описания внутреннего состояния. Эта функция не только интерпретирует результаты нового восприятия в свете существующих знаний о состоянии, но и использует информацию о том, как изменяется мир, для слежения за невидимыми частями мира, поэтому должна учитывать информацию о том, как действия агента влияют на состояние мира. Более подробные примеры приведены в главах 10 и 17.

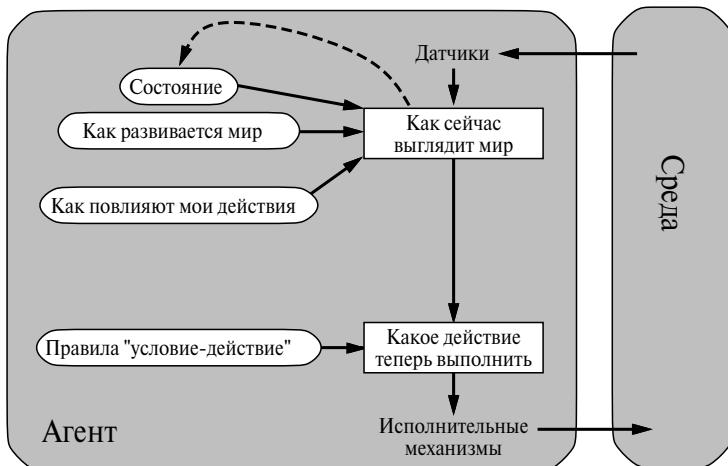


Рис. 2.4. Рефлексный агент, основанный на модели

**Листинг 2.4.** Рефлексный агент, основанный на модели, который следит за текущим состоянием мира с использованием внутренней модели, затем выбирает действие таким же образом, как и простой рефлексный агент

```

function Reflex-Agent-With-State(percept) returns действие action
    static: state, описание текущего состояния мира
            rules, множество правил условие–действие
            action, последнее по времени действие;
            первоначально не определено

    state  $\leftarrow$  Update-State(state, action, percept)
    rule  $\leftarrow$  Rule-Match(state, rules)
    action  $\leftarrow$  Rule-Action[rule]
    return action

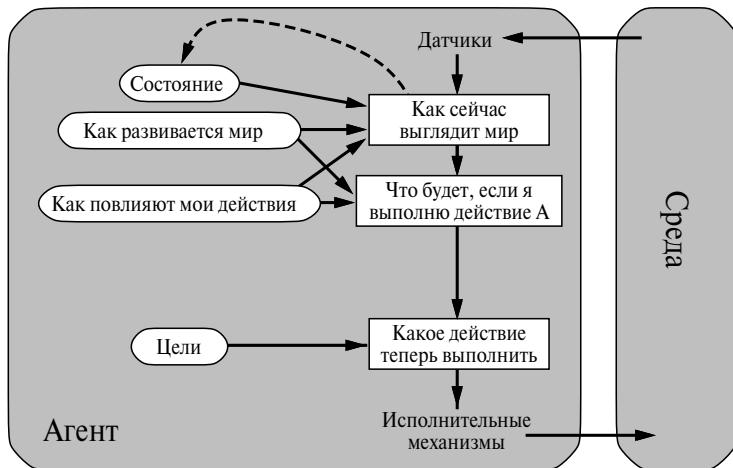
```

## Агенты, основанные на цели

Знаний о текущем состоянии среды не всегда достаточно для принятия решения о том, что делать. Например, на перекрестке дорог такси может повернуть налево, повернуть направо или ехать прямо. Правильное решение зависит от того, куда должно попасть это такси. Иными словами, агенту требуется не только описание текущего состояния, но и своего рода информация о **цели**, которая описывает желаемые ситуации, такие как доставка пассажира в место назначения. Программа агента может комбинировать эту информацию с информацией о результатах возможных действий (с такой же информацией, как и та, что использовалась при обновлении внутреннего состояния рефлексного агента) для выбора действий, позволяющих достичь этой цели. Структура агента, действующего на основе цели, показана на рис. 2.5.

Иногда задача выбора действия на основе цели решается просто, когда достижение цели немедленно становится результатом единственного действия, а иногда эта задача становится более сложной, и агенту требуется рассмотреть длинные последовательности движений и поворотов, чтобы найти способ достижения цели. Подобластиами искусственного интеллекта, посвященными выработке последовательно-

стей действий, позволяющих агенту достичь его целей, являются **поиск** (главы 3–6) и **планирование** (главы 11 и 12).



*Рис. 2.5. Агент, основанный на модели и на цели. Он следит за состоянием мира, а также за множеством целей, которых он пытается достичь, и выбирает действие, позволяющее (в конечном итоге) добиться достижения этих целей*

Следует учитывать, что процедура принятия решений такого рода имеет фундаментальные отличия от описанной выше процедуры применения правил условия–действие, поскольку в ней приходится размышлять о будущем, отвечая на два вопроса: “Что произойдет, если я сделаю то-то и то-то?” и “Позволит ли это мне достичь удовлетворения?” В проектах рефлексных агентов такая информация не представлена явно, поскольку встроенные правила устанавливают непосредственное соответствие между восприятиями и действиями. Рефлексный агент тормозит, увидев сигналы торможения движущегося впереди автомобиля, а агент, основанный на цели, может рассудить, что если на движущемся впереди автомобиле загорелись тормозные огни, то он замедляет свое движение. Учитывая принцип, по которому обычно изменяется этот мир, для него единственным действием, позволяющим достичь такой цели, как предотвращение столкновения с другими автомобилями, является торможение.

Хотя на первый взгляд кажется, что агент, основанный на цели, менее эффективен, он является более гибким, поскольку знания, на которые опираются его решения, представлены явно и могут быть модифицированы. Если начинается дождь, агент может обновить свои знания о том, насколько эффективно теперь будут работать его тормоза; это автоматически вызывает изменение всех соответствующих правил поведения с учетом новых условий. Для рефлексного агента, с другой стороны, в таком случае пришлось бы переписать целый ряд правил условия–действие. Поведение агента, основанного на цели, можно легко изменить, чтобы направить его в другое место, а правила рефлексного агента, которые указывают, где поворачивать и где ехать прямо, окажутся применимыми только для единственного места назначения; для того чтобы этого агента можно было направить в другое место, все эти правила должны быть заменены.

## Агенты, основанные на полезности

В действительности в большинстве вариантов среды для выработки высококачественного поведения одного лишь учета целей недостаточно. Например, обычно существует много последовательностей действий, позволяющих такси добраться до места назначения (и тем самым достичь поставленной цели), но некоторые из этих последовательностей обеспечивают более быструю, безопасную, надежную или недорогую поездку, чем другие. Цели позволяют провести лишь жесткое бинарное различие между состояниями “удовлетворенности” и “неудовлетворенности”, тогда как более общие показатели производительности должны обеспечивать сравнение различных состояний мира в точном соответствии с тем, насколько удовлетворенным станет агент, если их удастся достичь. Поскольку понятие “удовлетворенности” представляется не совсем научным, чаще применяется терминология, согласно которой состояние мира, более предпочтительное по сравнению с другим, рассматривается как имеющее более высокую **полезность** для агента<sup>8</sup>.

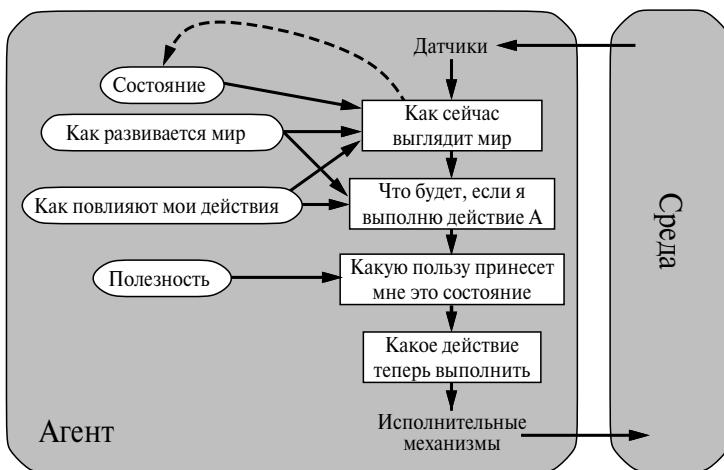
**Функция полезности** отображает состояние (или последовательность состояний) на вещественное число, которое обозначает соответствующую степень удовлетворенности агента. Полная спецификация функции полезности обеспечивает возможность принимать рациональные решения в описанных ниже двух случаях, когда этого не позволяют сделать цели. Во-первых, если имеются конфликтующие цели, такие, что могут быть достигнуты только некоторые из них (например, или скорость, или безопасность), то функция полезности позволяет найти приемлемый компромисс. Во-вторых, если имеется несколько целей, к которым может стремиться агент, но ни одна из них не может быть достигнута со всей определенностью, то функция полезности предоставляет удобный способ взвешенной оценки вероятности успеха с учетом важности целей.

В главе 16 будет показано, что любой рациональный агент должен вести себя так, как если бы он обладал функцией полезности, ожидаемое значение которой он пытается максимизировать. Поэтому агент, обладающий явно заданной функцией полезности, имеет возможность принимать рациональные решения и способен делать это с помощью алгоритма общего назначения, не зависящего от конкретной максимизируемой функции полезности. Благодаря этому “глобальное” определение рациональности (согласно которому рациональными считаются функции агента, имеющие наивысшую производительность) преобразуется в “локальное” ограничение на проекты рациональных агентов, которое может быть выражено в виде простой программы.

Структура агента, действующего с учетом полезности, показана на рис. 2.6. Программы агентов, действующих с учетом полезности, приведены в части V, которая посвящена проектированию агентов, принимающих решения, способных учитывать неопределенность, свойственную частично наблюдаемым вариантам среды.

---

<sup>8</sup> Термин “полезность” имеет англоязычный эквивалент “utility”, который в данном контексте обозначает “свойство быть полезным”, а не электростанцию или предприятие, предоставляющее коммунальные услуги.



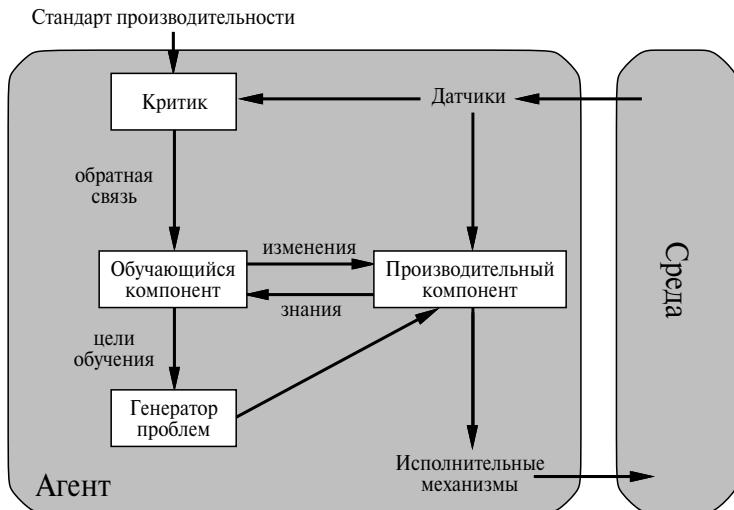
*Рис. 2.6. Агент, основанный на модели и на полезности. В нем модель мира используется наряду с функцией полезности, которая измеряет предпочтения агента применительно к состояниям мира. Затем агент выбирает действие, которое ведет к наилучшей ожидаемой полезности. Для вычисления ожидаемой полезности выполняется усреднение по всем возможным результирующим состояниям с учетом коэффициента, определяющего вероятность каждого результата*

## Обучающиеся агенты

Выше были описаны программы агентов, в которых применяются различные методы выбора действий. Но до сих пор еще не были приведены сведения о том, как создаются программы агентов. В своей знаменитой ранней статье Тьюринг [1520] проанализировал идею о том, как фактически должно осуществляться программирование предложенных им интеллектуальных машин вручную. Он оценил объем работы, который для этого потребуется, и пришел к такому выводу: “Желательно было бы иметь какой-то более продуктивный метод”. Предложенный им метод заключался в том, что необходимо создавать обучающиеся машины, а затем проводить их обучение. Теперь этот метод стал доминирующим методом создания наиболее современных систем во многих областях искусственного интеллекта. Как отмечалось выше, обучение имеет еще одно преимущество: оно позволяет агенту функционировать в первоначально неизвестных ему вариантах среды и становиться более компетентным по сравнению с тем, что могли бы позволить только его начальные знания. В данном разделе кратко представлены основные сведения об обучающихся агентах. Существующие возможности и методы обучения агентов конкретных типов рассматриваются почти в каждой главе данной книги, а в части VI более подробно описываются сами алгоритмы обучения.

Как показано на рис. 2.7, структура обучающегося агента может подразделяться на четыре концептуальных компонента. Наиболее важное различие наблюдается между **обучающим компонентом**, который отвечает за внесение усовершенствований, и **производительным компонентом**, который обеспечивает выбор внешних действий. Производительным компонентом является то, что до сих пор в данной книге рассматривалось в качестве всего агента: он получает воспринимаемую ин-

формацию и принимает решение о выполнении действий. Обучающий компонент использует информацию обратной связи от **критика** с оценкой того, как действует агент, и определяет, каким образом должен быть модифицирован производительный компонент для того, чтобы он успешнее действовал в будущем.



*Рис. 2.7. Общая модель обучающихся агентов*

Проект обучающего компонента во многом зависит от проекта производительного компонента. Осуществляя попытку спроектировать агента, который обучается определенным способностям, необходимо прежде всего стремиться найти ответ на вопрос: “Какого рода производительный компонент потребуется моему агенту после того, как он будет обучен тому, как выполнять свои функции?”, а не на вопрос: “Как приступить к решению задачи обучения его выполнению этих функций?” После того как спроектирован сам агент, можно приступить к конструированию обучающих механизмов, позволяющих усовершенствовать любую часть этого агента.

Критик сообщает обучающему компоненту, насколько хорошо действует агент с учетом постоянного стандарта производительности. Критик необходим, поскольку сами результаты восприятия не дают никаких указаний на то, успешно ли действует агент. Например, шахматная программа может получить результаты восприятия, указывающие на то, что она поставила мат своему противнику, но ей требуется стандарт производительности, который позволил бы определить, что это — хороший результат; сами данные восприятия ничего об этом не говорят. Важно, чтобы стандарт производительности был постоянным. В принципе этот стандарт следует рассматривать как полностью внешний по отношению к агенту, поскольку агент не должен иметь возможности его модифицировать так, чтобы он в большей степени соответствовал его собственному поведению.

Последним компонентом обучающегося агента является **генератор проблем**. Его задача состоит в том, чтобы предлагать действия, которые должны привести к получению нового и информативного опыта. Дело в том, что если производительный компонент предоставлен самому себе, то продолжает выполнять действия, которые являются наилучшими с точки зрения того, что он знает. Но если агент готов

к тому, чтобы немного поэкспериментировать и в кратковременной перспективе выполнять действия, которые, возможно, окажутся не совсем оптимальными, то он может обнаружить гораздо более лучшие действия с точки зрения долговременной перспективы. Генератор проблем предназначен именно для того, чтобы предлагать такие исследовательские действия. Именно этим занимаются ученые, проводя эксперименты. Галилей не считал, что сбрасывание камней с вершины Пизанской башни является самоцелью. Он не старался просто вдрызг разбить эти булыжники или оказать физическое воздействие на головы неудачливых прохожих. Его замысел состоял в том, чтобы изменить взгляды, сложившиеся в его собственной голове, сформулировав лучшую теорию движения объектов.

Для того чтобы перевести весь этот проект на конкретную почву, вернемся к примеру автоматизированного такси. Производительный компонент состоит из той коллекции знаний и процедур, которая применяется водителем такси при выборе им действий по вождению. Водитель такси с помощью этого производительного компонента выезжает на дорогу и ведет свою машину. Критик наблюдает за миром и в ходе этого передает соответствующую информацию обучающему компоненту. Например, после того как такси быстро выполняет поворот налево, пересекая три полосы движения, критик замечает, какие шокирующие выражения используют другие водители. На основании этого опыта обучающий компонент способен сформулировать правило, которое гласит, что это — недопустимое действие, а производительный компонент модифицируется путем установки нового правила. Генератор проблем может определить некоторые области поведения, требующие усовершенствования, и предложить эксперименты, такие как проверка тормозов на разных дорожных покрытиях и при различных условиях.

Обучающий компонент может вносить изменения в любой из компонентов “знаний”, показанных на схемах агентов (см. рис. 2.3–2.6). В простейших случаях обучение будет осуществляться непосредственно на основании последовательности актов восприятия. Наблюдение за парами последовательных состояний среды позволяет агенту освоить информацию о том, “как изменяется мир”, а наблюдение за результатами своих действий может дать агенту возможность узнать, “какое влияние оказывают мои действия”. Например, после того как водитель такси приложит определенное тормозное давление во время езды по мокрой дороге, он вскоре узнает, какое снижение скорости фактически было достигнуто. Очевидно, что эти две задачи обучения становятся более сложными, если среда наблюдаема лишь частично.

Те формы обучения, которые были описаны в предыдущем абзаце, не требуют доступа к внешнему стандарту производительности, вернее, в них применяется универсальный стандарт, согласно которому сделанные прогнозы должны быть согласованы с экспериментом. Ситуация становится немного сложнее, когда речь идет об агенте, основанном на полезности, который стремится освоить в процессе обучения информацию о полезности. Например, предположим, что агент, занимающийся вождением такси, перестает получать чаевые от пассажиров, которые в ходе утомительной поездки почувствовали себя полностью разбитыми. Внешний стандарт производительности должен информировать агента, что отсутствие чаевых — это отрицательный вклад в его общую производительность; в таком случае агент получает возможность освоить в результате обучения, что грубые маневры, утомляющие пассажиров, не позволяют повысить оценку его собственной функции полезности. В этом смысле стандарт производительности позволяет выделить определенную

часть входных результатов восприятия как **вознаграждение** (или **штраф**), непосредственно предоставляемое данными обратной связи, влияющими на качество поведения агента. Именно с этой точки зрения могут рассматриваться жестко закрепленные стандарты производительности, такие как боль или голод, которыми характеризуется жизнь животных. Эта тема рассматривается более подробно в главе 21.

Подводя итог, отметим, что агенты имеют самые различные компоненты, а сами эти компоненты могут быть представлены в программе агента многими способами, поэтому создается впечатление, что разнообразие методов обучения чрезвычайно велико. Тем не менее все эти методы имеют единый объединяющий их аспект. Процесс обучения, осуществляемый в интеллектуальных агентах, можно в целом охарактеризовать как процесс модификации каждого компонента агента для обеспечения более точного соответствия этих компонентов доступной информации обратной связи и тем самым улучшения общей производительности агента.

## 2.5. РЕЗЮМЕ

Эту главу можно сравнить с головокружительным полетом над обширным ландшафтом искусственного интеллекта, который мы рассматриваем как науку проектирования агентов. Ниже кратко приведены основные идеи, которые рассматривались в данной главе.

- **Агентом** является нечто воспринимающее и действующее в определенной среде. **Функция агента** определяет действие, предпринимаемое агентом в ответ на любую последовательность актов восприятия.
- **Показатели производительности** оценивают поведение агента в среде. **Рациональный агент** действует так, чтобы максимизировать ожидаемые значения показателей производительности, с учетом последовательности актов восприятия, полученной агентом к данному моменту.
- Спецификация **проблемной среды** включает определения показателей производительности, внешней среды, исполнительных механизмов и датчиков. Первым этапом проектирования агента всегда должно быть определение проблемной среды с наибольшей возможной полнотой.
- Варианты проблемной среды классифицируются по нескольким важным размерностям. Они могут быть полностью или частично наблюдаемыми, детерминированными или стохастическими, эпизодическими или последовательными, статическими или динамическими, дискретными или непрерывными, а также одноагентными или мультиагентными.
- **Программа агента** реализует функцию агента. Существует целый ряд основных проектов программ агента, соответствующих характеру явно воспринимаемой информации, которая используется в процессе принятия решения. Разные проекты характеризуются различной эффективностью, компактностью и гибкостью. Выбор наиболее подходящего проекта программы агента зависит от характера среды.
- **Простые рефлексные агенты** отвечают непосредственно на акты восприятия, тогда как **рефлексные агенты, основанные на модели**, поддерживают внутреннее

состояние, прослеживая те аспекты среды, которые не наблюдаются в текущем акте восприятия. **Агенты, действующие на основе цели**, организуют свои действия так, чтобы достичнуть своих целей, а **агенты, действующие с учетом полезности**, пытаются максимизировать свою собственную ожидаемую “удовлетворенность”.

- Все агенты способны улучшать свою работу благодаря **обучению**.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Истоки представлений о центральной роли действий в интеллекте (сформулированных в виде понятия *практических рассуждений*) прослеживаются до *Никомаховой этики* Аристотеля. Практические рассуждения были также темой влиятельной статьи Маккарти *Programs with Common Sense* (Программы со здравым смыслом) [1009]. Такие области науки, как робототехника и теория управления, по самому своему характеру преимущественно касаются проблем конструирования физических агентов. Понятие **контроллера** в теории управления идентично понятию агента в искусственном интеллекте. И хотя на первый взгляд это может показаться удивительным, но на протяжении большей части истории развития искусственного интеллекта основные усилия в этой области были сосредоточены на исследовании отдельных компонентов агентов (в качестве примеров можно привести системы поиска ответов на вопросы, программы автоматического доказательства теорем, системы технического зрения и т.д.), а не самих агентов, рассматриваемых как единое целое. Важным исключением из этого правила, оказавшим значительное влияние на дальнейшее развитие данной области, стало обсуждение проблематики агентов в работе Генезерета и Нильссона [537]. В настоящее время в данной научной области широко применяется подход, основанный на изучении всего агента, и результаты, достигнутые в рамках этого подхода, стали центральной темой новейших работ [1146], [1227].

В главе 1 было показано, что корни понятия *рациональности* прослеживаются в философии и экономике. В искусственном интеллекте это понятие не привлекало значительного интереса до тех пор, пока в середине 1980-х не началось широкое обсуждение проблемы создания подходящих теоретических основ данной области. В статье Джона Дойла [410] было предсказано, что со временем проектирование рациональных агентов станет рассматриваться в качестве основного назначения искусственного интеллекта, притом что другие популярные ныне темы исследований послужат основой формирования новых дисциплин.

Стремление к тщательному изучению свойств среды и их влияния на выбор самого подходящего проекта рационального агента наиболее ярко проявляется в традиционных областях теории управления; например, в исследованиях по классическим системам управления [405] рассматриваются полностью наблюдаемые, детерминированные варианты среды; темой работ по стохастическому оптимальному управлению [865] являются частично наблюдаемые, стохастические варианты среды; а работы по гибридному управлению [649] касаются таких вариантов среды, которые содержат и дискретные, и непрерывные элементы. Описание различий между полностью и частично наблюдаемыми вариантами среды является также центральной темой работ по **динамическому программированию**, проводимых в области исследования операций [1244], которая будет обсуждаться в главе 17.

Рефлексные агенты были основной моделью для психологических бихевиористов, таких как Скиннер [1423], которые пытались свести все знания в области психологии организмов исключительно к отображениям “ввод–вывод” или “стимул–отклик”. В результате прогресса в области психологии, связанного с переходом от бихевиоризма к функционализму, который был по крайней мере отчасти обусловлен распространением на агентов трактовки понятия компьютера как новой метафоры в мировоззрении человека [923], [1246], возникло понимание того, что нужно также учитывать внутреннее состояние агента. В большинстве работ по искусственному интеллекту идея чисто рефлексных агентов с внутренним состоянием рассматривалась как слишком простая для того, чтобы на ее основе можно было добиться значительных успехов, но исследования Розеншайна [1308] и Брукса [189] позволили поставить под сомнение это предположение (см. главу 25). В последние годы значительная часть работ была посвящена поиску эффективных алгоритмов слежения за сложными вариантами среды [610]. Наиболее впечатляющим примером достигнутых при этом результатов является программа *Remote Agent*, которая управляла космическим аппаратом Deep Space One (описанная на с. 69) [744], [1108].

Понимание необходимости создания агентов на основе цели просматривается во всем, что стало источником идей искусственного интеллекта, начиная с введенного Аристотелем определения практического рассуждения и заканчивая ранними статьями Маккарти по логическому искусственному интеллекту. Робот *Shakey* [466], [1143] был первым воплощением логического агента на основе цели в виде автоматизированного устройства. Полный логический анализ агентов на основе цели приведен в книге Генезерета и Нильссона [537], а методология программирования на основе цели, получившая название *агентно-ориентированного программирования*, была разработана Шохемом [1404].

Кроме того, начиная с книги *Human Problem Solving* [1130], оказавшей чрезвычайно большое влияние на ход дальнейших исследований, в той области когнитивной психологии, которая посвящена изучению процедур решения задач человеком, ведущее место занял подход, основанный на понятии цели; на этом подходе базируется также вся последняя работа Ньюэлла [1125]. Цели, дополнительно подразделяемые на желания (общие замыслы) и намерения (осуществляемые в настоящее время), являются основой теории агентов, разработанной Братманом [176]. Эта теория оказала влияние и на работы в области понимания естественного языка, и на исследования мультиагентных систем.

Горвиц, наряду с другими исследователями [686], особо подчеркивал важность использования в качестве основы для искусственного интеллекта понятия *рациональности*, рассматриваемой как средство максимизации ожидаемой полезности. Книга Перла [1191] была первой работой в области искусственного интеллекта, в которой дан глубокий анализ проблем применения теории вероятности и теории полезности; описанные им практические методы проведения рассуждений и принятия решений в условиях неопределенности, по-видимому, послужили наиболее важной причиной быстрой смены направления исследований в 1990-х годах и перехода к изучению агентов, действующих с учетом полезности (подробнее об этом рассказывается в части V).

Общий проект для обучающихся агентов, приведенный на рис. 2.7, является классическим образцом такого проекта в литературе по машинному обучению [203], [1064]. Одним из первых примеров воплощения этого проекта в программах является

ся обучающаяся программа для игры в шашки Артура Самюэла [1349], [1350]. Обучающиеся агенты подробно рассматриваются в части VI.

В последние годы быстро растет интерес к агентам и к проектам агентов, отчасти в связи с расширением Internet и осознанием необходимости создания автоматизированных и мобильных **программных роботов** [447]. Сборники статей по этой теме можно найти в работах *Readings in Agents* [704] и *Foundations of Rational Agency* [1615]. В книге *Multiagent Systems* [1564] представлены надежные теоретические основы для многих аспектов проектирования агентов. К числу конференций, посвященных агентам, относятся *International Conference on Autonomous Agents*, *International Workshop on Agent Theories, Architectures, and Languages* и *International Conference on Multiagent Systems*. И наконец отметим, что в книге *Dung Beetle Ecology* [615] приведен большой объем интересной информации о поведении навозных жуков (о котором говорилось в данной главе).

## УПРАЖНЕНИЯ

---

- 2.1.** Самостоятельно сформулируйте определения следующих понятий: агент; функция агента; программа агента; рациональность; автономность; рефлексный агент; агент, основанный на модели; агент на основе цели; агент на основе полезности; обучающийся агент.
- 2.2.** Для измерения того, насколько успешно функционирует агент, используются и показатели производительности, и функция полезности. Объясните, в чем состоит различие между этими двумя критериями.
- 2.3.** В этом упражнении исследуются различия между функциями агента и программами агента.
  - a)** Может ли существовать больше чем одна программа агента, которая реализует данную функцию агента? Приведите пример, подтверждающий положительный ответ, или покажите, почему такая ситуация невозможна.
  - б)** Есть ли такие функции агента, которые не могут быть реализованы никакими программами агента?
  - в)** Верно ли, что каждая программа агента реализует точно одну функцию агента, при условии, что архитектура вычислительной машины остается неизменной?
  - г)** Если в архитектуре предусмотрено  $n$  битов памяти, то сколько различных возможных программ агента может быть реализовано с ее помощью?
- 2.4.** В этом упражнении исследуется рациональность различных функций агента для пылесоса.
  - а)** Покажите, что простая функция агента-пылесоса, описанная в табл. 2.1, действительно рациональна, согласно предположениям, перечисленным на с. 79.
  - б)** Опишите рациональную функцию агента для модифицированного показателя производительности, который предусматривает штраф в один пункт за каждое движение. Требуется ли поддержка внутреннего состояния для соответствующей программы агента?

- в) Обсудите возможные проекты агентов для случаев, в которых чистые квадраты могут стать грязными, а география среды неизвестна. Имеет ли смысл в таких случаях для агента обучаться на своем опыте? Если да, то что он должен изучать?
- 2.5. Для каждого из следующих агентов разработайте описание PEAS среды задачи:
- робот-футболист;
  - агент, совершающий покупки книг в Internet;
  - автономный марсианский вездеход;
  - ассистент математика, занимающийся доказательством теорем.
- 2.6. Для каждого из типов агентов, перечисленных в упр. 2.5, охарактеризуйте среду в соответствии со свойствами, приведенными в разделе 2.3, и выберите подходящий проект агента.
- ▣ Все приведенные ниже упражнения касаются реализации вариантов среды и агентов для мира пылесоса.
- 2.7. Реализуйте имитатор среды измерения производительности для мира пылесоса, который изображен на рис. 2.2 и определен на с. 79. Предложенная реализация должна быть модульной, для того чтобы можно было заменять датчики и исполнительные механизмы, а также модифицировать характеристики среды (размер, форму, размещение мусора и т.д.). (*Примечание.* Для некоторых сочетаний языка программирования и операционной системы в оперативном репозитарии кода уже имеются реализации.)
- 2.8. Реализуйте простого рефлексного агента для среды пылесоса, которая рассматривается в упр. 2.7. Вызовите на выполнение имитатор среды с этим агентом для всех возможных начальных конфигураций мусора и местоположений агента. Зарегистрируйте оценки производительности работы агента для каждой конфигурации и определите его общую среднюю оценку.
- 2.9. Рассмотрите модифицированную версию среды пылесоса из упр. 2.7, в которой агенту назначается штраф в один пункт за каждое движение.
- Может ли быть полностью рациональным простой рефлексный агент для этой среды? Обоснуйте свой ответ.
  - А что можно сказать о рефлексном агенте с внутренним состоянием? Спроектируйте такого агента.
  - Как изменятся приведенные вами ответы на вопросы а) и б), если акты восприятия агента позволяют ему иметь информацию о том, является ли чистым или грязным каждый квадрат в этой среде?
- 2.10. Рассмотрите модифицированную версию среды пылесоса из упр. 2.7, в которой неизвестны география среды (ее протяженность, границы и препятствия), а также начальная конфигурация расположения мусора. (Агент может совершать движения *Left* и *Right*, а также *Up* и *Down*.)
- Может ли быть полностью рациональным простой рефлексный агент для этой среды? Обоснуйте свой ответ.
  - Может ли простой рефлексный агент с рандомизированной функцией агента превзойти по своей производительности простого рефлексного

агента? Спроектируйте такого агента и измерьте его производительность в нескольких вариантах среды.

- в) Можете ли вы спроектировать среду, в которой предложенный вами рандомизированный агент будет показывать очень низкую производительность? Продемонстрируйте полученные вами результаты.
  - г) Может ли рефлексный агент с поддержкой состояния превзойти по своей производительности простого рефлексного агента? Спроектируйте такого агента и измерьте его производительность в нескольких вариантах среды. Сумеете ли вы спроектировать рационального агента этого типа?
- 2.11.** Повторите упр. 2.10 для такого случая, в котором датчик местоположения заменен датчиком удара, позволяющим обнаруживать попытки агента пройти сквозь препятствие или пересечь границы среды. Предположим, что датчик удара перестал работать; как должен действовать агент?
- 2.12.** Все варианты среды пылесоса, рассматриваемые в предыдущих упражнениях, были детерминированными. Обсудите возможные программы агента для каждого из следующих стохастических вариантов.
- а) Закон Мэрфи: в течение 25% времени применение действия *Suck* не позволяет очистить пол, если пол грязный, и приводит к появлению мусора на полу, если пол чистый. Как эти обстоятельства отразятся на предложенной вами программе агента, если датчик мусора дает неправильный ответ в течение 10% времени?
  - б) Маленькие дети: в каждом интервале времени каждый чистый квадрат имеет 10%-ную вероятность стать грязным. Можете ли вы предложить проект рационального агента для этого случая?

## Часть II

# РЕШЕНИЕ ПРОБЛЕМ

Решение проблем посредством поиска	110
Информированный поиск и исследование пространства состояний	153
Задачи удовлетворения ограничений	209
Поиск в условиях противодействия	240

# 3 РЕШЕНИЕ ПРОБЛЕМ ПОСРЕДСТВОМ ПОИСКА

*В этой главе показано, каким образом агент может найти последовательность действий, позволяющую достичь его целей в тех условиях, когда единственного действия для этого недостаточно.*

Простейшими агентами, которые рассматривались в главе 2, были рефлексные агенты, функционирование которых основано на непосредственном отображении состояний в действия. Подобные агенты не могут успешно действовать в тех вариантах среды, где такие отображения были бы слишком большими, чтобы можно было обеспечить их хранение, а изучение отображений потребовало бы слишком много времени. Агенты на основе цели, с другой стороны, способны достичь успеха, рассматривая будущие действия и оценивая желательность их результатов.

Данная глава посвящена описанию одной разновидности агента на основе цели, называемой **агентом, решающим задачи**. Агенты, решающие задачи, определяют, что делать, находя последовательности действий, которые ведут к желаемым состояниям. Начнем изложение этой темы с точного определения элементов, из которых состоит “задача” и ее “решение”, и приведем несколько примеров для иллюстрации этих определений. Затем представим несколько алгоритмов поиска общего назначения, которые могут использоваться для решения подобных задач, и проведем сравнение преимуществ и недостатков каждого алгоритма. Эти алгоритмы являются **неинформированными** в том смысле, что в них не используется какая-либо информация о рассматриваемой задаче, кроме ее определения. В главе 4 речь идет об **информированных** алгоритмах поиска, в которых используются определенные сведения о том, где следует искать решения.

В данной главе применяются понятия из области анализа алгоритмов. Читатели, незнакомые с понятиями асимптотической сложности (т.е. с системой обозначений  $O()$ ) и NP-полноты, должны обратиться к приложению A.

## 3.1. АГЕНТЫ, РЕШАЮЩИЕ ЗАДАЧИ

Предполагается, что интеллектуальные агенты обладают способностью максимизировать свои показатели производительности. Как уже упоминалось в главе 2, реа-

лизация указанного свойства в определенной степени упрощается, если агент способен принять на себя обязанность достичь **цели** и стремиться к ее удовлетворению. Вначале рассмотрим, как и почему агент может приобрести такую способность.

Представьте себе, что некоторый агент находится в городе Арад, Румыния, и проводит свой отпуск в качестве туриста. Показатели производительности агента включают много компонентов: он хочет улучшить свой загар, усовершенствовать знание румынского языка, осмотреть достопримечательности, насладитьсяочной жизнью (со всеми ее привлекательными сторонами), избежать неприятностей и т.д. Эта задача принятия решения является сложной; для ее выполнения необходимо учитывать множество компромиссов и внимательно читать путеводители. Кроме того, предположим, что у агента имеется не подлежащий возмещению билет для вылета из Бухареста на следующий день. В данном случае для агента имеет смысл стремиться к достижению **цели** попасть в Бухарест. Способы действий, не позволяющие вовремя попасть в Бухарест, могут быть отвергнуты без дальнейшего рассмотрения и поэтому задача принятия решения агентом значительно упрощается. Цели позволяют организовать поведение, ограничивая выбор промежуточных этапов, которые пытается осуществить агент. Первым шагом в решении задачи является **формулировка цели** с учетом текущей ситуации и показателей производительности агента.

Мы будем рассматривать цель как множество состояний мира, а именно тех состояний, в которых достигается такая цель. Задача агента состоит в том, чтобы определить, какая последовательность действий приведет его в целевое состояние. Прежде чем это сделать, агент должен определить, какого рода действия и состояния ему необходимо рассмотреть. Но если бы агент пытался рассматривать действия на уровне “перемещения левой ноги вперед на один сантиметр” или “поворота рулевого колеса на один градус влево”, то, по-видимому, так и не смог бы выехать с автомобильной стоянки, не говоря уже о своевременном прибытии в Бухарест, поскольку на таком уровне детализации мир обладает слишком большой неопределенностью, а решение должно состоять из слишком многих этапов. **Формулировка задачи** представляет собой процесс определения того, какие действия и состояния следует рассматривать с учетом некоторой цели. Этот процесс будет описан более подробно немного позднее. А на данный момент предположим, что агент будет рассматривать действия на уровне автомобильной поездки из одного крупного города в другой. Таким образом, состояния, рассматриваемые агентом, соответствуют его пребыванию в конкретном городе<sup>1</sup>.

Итак, наш агент поставил перед собой цель доехать на автомобиле до Бухареста и определяет, куда отправиться для этого из Арада. Из этого города ведут три дороги: в Сибиу, Тимишоару и Зеринд. Прибытие в какой-либо из этих городов не представляет собой достижение намеченной цели, поэтому агент, не очень знакомый с географией Румынии, не будет знать, по какой дороге он должен следовать<sup>2</sup>. Иными словами, агент не знает, какое из его возможных действий является наилучшим, поскольку не обладает достаточными знаниями о состоянии, возникающем в результа-

<sup>1</sup> Обратите внимание на то, что каждое из этих “состояний” фактически соответствует большому множеству состояний мира, поскольку в состоянии реального мира должен быть определен каждый аспект действительности. Важно всегда учитывать различие между состояниями, которые рассматриваются в проблемной области решения задач, и состояниями реального мира.

<sup>2</sup> Авторы предполагают, что большинство читателей окажутся в таком же положении и вполне могут представить себя такими же беспомощными, как и наш агент. Приносим свои извинения румынским читателям, которые не смогут воспользоваться преимуществами использованного здесь педагогического приема.

те выполнения каждого действия. Если агент не получит дополнительных знаний, то окажется в тупике. Самое лучшее, что он может сделать, — это выбрать одно из указанных действий случайным образом.

Но предположим, что у агента есть карта Румынии, либо на бумаге, либо в его памяти. Карта способна обеспечить агента информацией о состояниях, в которых он может оказаться, и о действиях, которые он способен предпринять. Агент имеет возможность воспользоваться этой информацией для определения последовательных этапов гипотетического путешествия через каждый из этих трех городов в попытке найти такой путь, который в конечном итоге приведет его в Бухарест. Найдя на карте путь от Арада до Бухареста, агент может достичь своей цели, осуществляя действия по вождению автомобиля, которые соответствуют этапам этого путешествия. Вообще говоря, *агент, имеющий несколько непосредственных вариантов выбора с неизвестной стоимостью, может решить, что делать, исследуя вначале различные возможные последовательности действий, которые ведут к состояниям с известной стоимостью, а затем выбирая из них наилучшую последовательность.*

Описанный процесс определения такой последовательности называется **поиском**. Любой алгоритм поиска принимает в качестве входных данных некоторую задачу и возвращает **решение** в форме последовательности действий. После того как решение найдено, могут быть осуществлены действия, рекомендованные этим алгоритмом. Такое осуществление происходит на стадии **выполнения**. Итак, для рассматриваемого агента можно применить простой проект, позволяющий применить принцип “сформулировать, найти, выполнить”, как показано в листинге 3.1. После формулировки цели и решаемой задачи агент вызывает процедуру поиска для решения этой задачи. Затем он использует полученное решение для руководства своими действиями, выполняя в качестве следующего предпринимаемого мероприятия все, что рекомендовано в решении (как правило, первым является первое действие последовательности), а затем удаляет этот этап из последовательности. Сразу после выполнения этого решения агент формулирует новую цель.

**Листинг 3.1. Простой агент, решающий задачу.** Вначале он формулирует цель и задачу, затем ищет последовательность действий, позволяющую решить эту задачу, и, наконец, осуществляет действия одно за другим. Закончив свою работу, агент формулирует другую цель и начинает сначала. Обратите внимание на то, что при выполнении очередной последовательности действий агент игнорирует данные своих актов восприятия, поскольку предполагает, что найденное им решение всегда выполнимо

---

```

function Simple-Problem-Solving-Agent(percept) returns действие action
  inputs: percept, результат восприятия
  static: seq, последовательность действий, первоначально пустая
           state, некоторое описание текущего состояния мира
           goal, цель, первоначально неопределенная
           problem, формулировка задачи

  state  $\leftarrow$  Update-State(state, percept)
  if последовательность seq пуста then do
    goal  $\leftarrow$  Formulate-Goal(state)
    problem  $\leftarrow$  Formulate-Problem(state, goal)
    seq  $\leftarrow$  Search(problem)
  action  $\leftarrow$  First(seq)
  seq  $\leftarrow$  Rest(seq)
  return action

```

---

Вначале опишем процесс составления формулировки задачи, а затем посвятим основную часть данной главы рассмотрению различных алгоритмов для функции Search. В этой главе осуществление функций Update-State и Formulate-Goal дополнительно не рассматривается.

Прежде чем перейти к подробному описанию, сделаем краткую паузу, чтобы определить, в чем агенты, решающие задачи, соответствуют описанию агентов и вариантов среды, приведенному в главе 2. В проекте агента, показанном в листинге 3.1, предполагается, что данная среда является **статической**, поскольку этапы формулировки и решения задачи осуществляются без учета каких-либо изменений, которые могут произойти в данной среде. Кроме того, в этом проекте агента допускается, что начальное состояние известно; получить такие сведения проще всего, если среда является **наблюдаемой**. К тому же в самой идее перечисления “альтернативных стратегий” заключается предположение, что среда может рассматриваться как **дискретная**. Последней и наиболее важной особенностью является следующее: в проекте агента предполагается, что среда является **детерминированной**. Решения задач представляют собой единственные последовательности действий, поэтому в них не могут учитываться какие-либо неожиданные события; более того, решения выполняются без учета результатов каких-либо актов восприятия! Агент, выполняющий свои планы, образно говоря, с закрытыми глазами, должен быть вполне уверенным в том, что он делает. (В теории управления подобный проект именуется системой **с разомкнутой обратной связью**, поскольку игнорирование результатов восприятия приводит к нарушению обратной связи между агентом и средой.) Все эти предположения означают, что мы имеем дело с простейшими разновидностями среды, и в этом состоит одна из причин, по которой настоящая глава помещена в самом начале данной книги. В разделе 3.6 дано краткое описание того, что произойдет, если будут исключены допущения о наблюдаемости и детерминированности, а в главах 12 и 17 приведено гораздо более подробное изложение соответствующей темы.

## Хорошо структурированные задачи и решения

❖ **Задача** может быть формально определена с помощью четырех *компонентов*, описанных ниже.

- ❖ **Начальное состояние**, в котором агент приступает к работе. Например, начальное состояние для нашего агента в Румынии может быть описано как пребывание в Араде, *In(Arad)*.
- Описание возможных действий, доступных агенту. В наиболее общей формулировке<sup>3</sup> используется **функция определения преемника**. Если задано конкретное состояние *x*, то функция Successor-Fn(*x*) возвращает множество упорядоченных пар *<action, successor>*, где каждое действие *action* представляет собой одно из допустимых действий в состоянии *x*, а каждый преемник *successor* представляет собой состояние, которое может быть достигнуто из *x* путем применения этого действия. Например, из состояния

<sup>3</sup> В альтернативной формулировке используется множество **операторов**, которые могут быть применены к некоторому состоянию для выработки преемников.

*In(Arad)* функция определения преемника для данной задачи проезда по Румынии возвратит следующее:

```
{<Go(Sibiu), In(Sibiu)>, <Go(Timisoara), In(Timisoara)>,
<Go(Zerind), In(Zerind)>}
```

Начальное состояние и функция определения преемника, вместе взятые, неявно задают пространство состояний данной задачи — множество всех состояний, достижимых из начального состояния. Пространство состояний образует граф, узлами которого являются состояния, а дугами между узлами — действия. (Карта Румынии, показанная на рис. 3.1, может интерпретироваться как граф пространства состояний, если каждая дорога рассматривается как обозначающая два действия проезда на автомобиле, по одному в каждом направлении.) Путем в пространстве состояний является последовательность состояний, соединенных последовательностью действий.

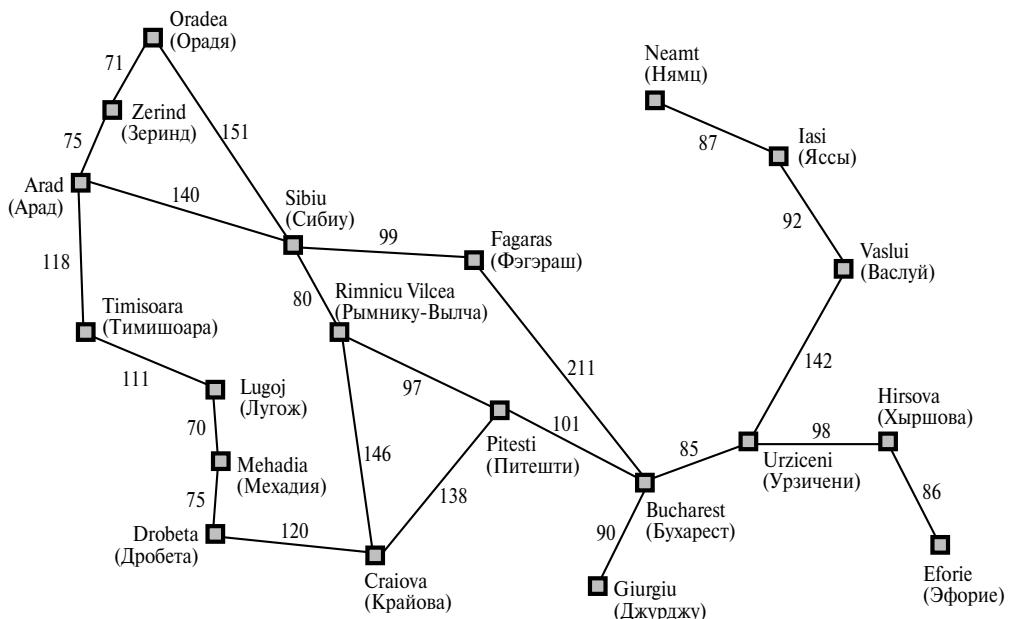


Рис. 3.1. Упрощенная дорожная карта части Румынии

- Проверка цели, позволяющая определить, является ли данное конкретное состояние целевым состоянием. Иногда имеется явно заданное множество возможных целевых состояний, и эта проверка сводится просто к определению того, является ли данное состояние одним из них. Цель рассматриваемого агента в Румынии представляет собой одноэлементное множество `{In(Bucharest)}`. Иногда цель задается в виде абстрактного свойства, а не явно перечисленного множества состояний. Например, в шахматах цель состоит в достижении состояния, называемого “матом”, в котором король противника атакован и не может уклониться от удара.
- Функция стоимости пути, которая назначает числовое значение стоимости каждому пути. Агент, решающий задачу, выбирает функцию стоимости, кото-

рая соответствует его собственным показателям производительности. Для данного агента, пытающегося попасть в Бухарест, важнее всего время, поэтому стоимость пути может измеряться длиной в километрах. В настоящей главе предполагается, что стоимость пути может быть описана как сумма стоимостей отдельных действий, выполняемых вдоль этого пути.  $\bowtie$  **Стоимость этапа**, связанного с выполнением действия  $a$  для перехода из состояния  $x$  в состояние  $y$ , обозначается как  $c(x, a, y)$ . Стоимости этапов для Румынии показаны на рис. 3.1 в виде дорожных расстояний. Предполагается, что стоимости этапов являются неотрицательными<sup>4</sup>.

Описанные выше элементы определяют задачу и могут быть собраны вместе в единую структуру данных, которая передается в качестве входных данных в алгоритм решения задачи. **Решением** задачи является путь от начального состояния до целевого состояния. Качество решения измеряется с помощью функции стоимости пути, а  $\bowtie$  **оптимальным решением** является такое решение, которое имеет наименьшую стоимость пути среди всех прочих решений.

### Формулировка задачи

В предыдущем разделе была предложена формулировка задачи переезда в Бухарест в терминах начального состояния, функции определения преемника, проверки цели и стоимости пути. Эта формулировка кажется приемлемой, но в ней все же не учитываются слишком многие аспекты реального мира. Сравним простое выбранное нами описание состояния,  $In(Arad)$ , с действительным путешествием по стране, в котором состояние мира должно учитывать так много факторов: попутчиков, текущие радиопередачи, вид из окна, наличие поблизости представителей сил правопорядка, расстояние до ближайшей остановки на отдых, состояние дороги, погоду и т.д. Все эти соображения исключены из наших описаний состояния, поскольку они не имеют отношения к задаче поиска маршрута поездки в Бухарест. Процесс удаления деталей из представления называется  $\bowtie$  **абстрагированием**.

Кроме абстрагирования описаний состояний, необходимо абстрагировать сами действия. Любое действие по рождению влечет за собой много следствий. Такие действия не только изменяют местонахождение автомобиля и его пассажиров, но и занимают время, приводят к потреблению топлива, влекут за собой загрязнение окружающей среды и влияют на самого агента (как говорится, путешествия расширяют кругозор). В нашей формулировке учитывается только изменение местонахождения. Кроме того, существует много действий, которые мы вообще не рассматриваем: включение радиоприемника, осмотр окрестностей из окна, замедление движения по требованию дорожной полиции и т.д. К тому же, безусловно, действия здесь не задаются на уровне “поворота рулевого колеса влево на три градуса”.

Можно ли достичь большей точности определения приемлемого уровня абстракции? Будем считать, что выбранные нами абстрактные состояния и действия соответствуют большим множествам более подробных описаний состояния мира, а также более детализированным последовательностям действий. Теперь рассмотрим решение абстрактной задачи, например поиска пути от Арада до Бухареста через города Сибиу, Рымнику-Вылча и Питешти. Это абстрактное решение соответствует

<sup>4</sup> Последствия, связанные с применением отрицательных стоимостей, рассматриваются в упр. 3.17.

большому количеству более подробных инструкций по преодолению пути. Например, можно вести автомобиль между городами Сибиу и Рымнику-Вылча с включенным радиоприемником, а затем выключить его на всю оставшуюся часть путешествия. Абстракция является действительной, если мы можем преобразовать любое абстрактное решение в такое решение, которое подходит для более детально описанного мира; достаточным условием является то, что для любого детализированного состояния, которое представляет собой “пребывание в городе Арад”, существует детализированный путь в некоторое состояние, соответствующее “пребыванию в городе Сибиу”, и т.д. Абстракция является полезной, если осуществление каждого из действий, предусмотренных в решении, становится проще по сравнению с первоначальной задачей; в этом случае действия являются достаточно простыми для того, чтобы они могли быть выполнены без дальнейшего поиска или планирования со стороны обычного агента, занимающегося вождением автомобиля. Поэтому выбор хорошей абстракции сводится к исключению максимально возможного количества подробностей и вместе с тем к сохранению способности оставаться действительной и обеспечению того, чтобы абстрактные действия были легко осуществимы. Если бы не было возможности создавать полезные абстракции, то интеллектуальные агенты не могли бы успешно действовать в реальном мире.

## 3.2. ПРИМЕРЫ ЗАДАЧ

---

Описанный подход к решению задач был применен в очень многих вариантах экспериментальной среды. В этом разделе перечислены некоторые из наиболее известных примеров решения задач, которые подразделяются на два типа — упрощенные и реальные задачи. **Упрощенная задача** предназначена для иллюстрации или проверки различных методов решения задач. Ей может быть дано краткое, точное описание. Это означает, что такая задача может легко использоваться разными исследователями для сравнения производительности алгоритмов. **Реальной задачей** называется такая задача, решение которой действительно требуется людям. Как правило, такие задачи не имеют единого приемлемого для всех описания, но мы попытаемся показать, как обычно выглядят их формулировки.

### Упрощенные задачи

В качестве первого примера рассмотрим **мир пылесоса**, впервые представленный в главе 2 (см. рис. 2.2). Деятельность в этом мире можно сформулировать в качестве задачи, как описано ниже.

- **Состояния.** Агент находится в одном из двух местонахождений, в каждом из которых может присутствовать или не присутствовать мусор. Поэтому существует  $2 \times 2^2 = 8$  возможных состояний мира.
- **Начальное состояние.** В качестве начального состояния может быть назначено любое состояние.
- **Функция определения преемника.** Эта функция вырабатывает допустимые состояния, которые являются следствием попыток выполнения трех действий (*Left*, *Right* и *Suck*). Полное пространство состояний показано на рис. 3.2.

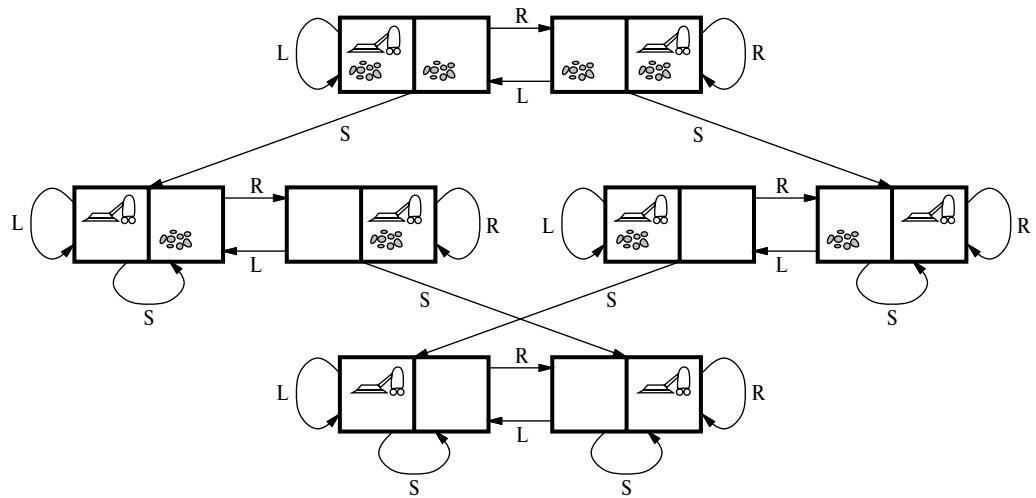


Рис. 3.2. Пространство состояний для мира пылесоса. Дуги обозначают действия: L=Left, R=Right, S=Suck

- **Проверка цели.** Эта проверка сводится к определению того, являются ли численными все квадраты.
- **Стоимость пути.** Стоимость каждого этапа равна 1, поэтому стоимость пути представляет собой количество этапов в этом пути.

По сравнению с задачей реального мира эта упрощенная задача характеризуется различимыми местонахождениями, возможностью определять наличие мусора, надежной очисткой, а также сохранением достигнутого состояния после очистки. (В разделе 3.6 некоторые из этих допущений будут исключены.) Необходимо учитывать, что состояние определяется и местонахождением агента, и наличием мусора. В более крупной среде с  $n$  местонахождениями имеется  $n \cdot 2^n$  состояний.

↗ **Задача игры в восемь**, экземпляр которой показан на рис. 3.3, состоит из доски  $3 \times 3$  с восемью пронумерованными фишками и с одним пустым участком. Фишка, смежная с пустым участком, может быть передвинута на этот участок. Требуется достичь указанного целевого состояния, подобного тому, которое показано в правой части рисунка. Стандартная формулировка этой задачи приведена ниже.

- **Состояния.** Описание состояния определяет местонахождение каждой из этих восьми фишек и пустого участка на одном из девяти квадратов.
- **Начальное состояние.** В качестве начального может быть определено любое состояние. Необходимо отметить, что любая заданная цель может быть достигнута точно из половины возможных начальных состояний (см. упр. 3.4).
- **Функция определения преемника.** Эта функция формирует допустимые состояния, которые являются результатом попыток осуществления указанных четырех действий (теоретически возможных ходов *Left*, *Right*, *Up* или *Down*).
- **Проверка цели.** Она позволяет определить, соответствует ли данное состояние целевой конфигурации, показанной на рис. 3.3. (Возможны также другие целевые конфигурации.)

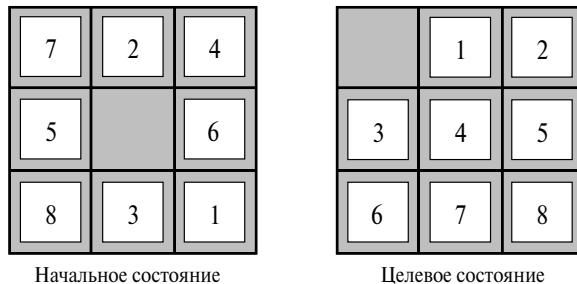


Рис. 3.3. Типичный экземпляр задачи игры в восемь

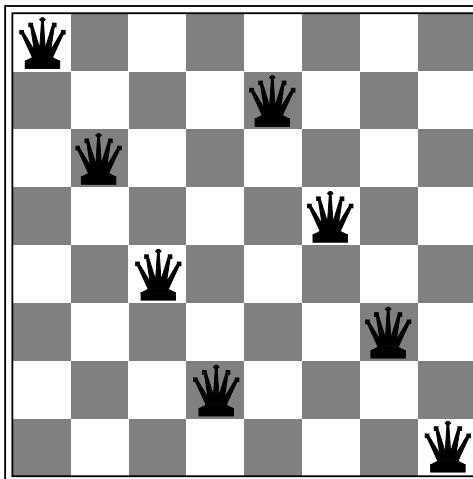
- **Стоимость пути.** Каждый этап имеет стоимость 1, поэтому стоимость пути равна количеству этапов в пути.

Какие абстракции здесь предусмотрены? Действия абстрагируются до уровня указания их начального и конечного состояний, при этом игнорируются промежуточные положения в процессе перемещения фишк. В ходе создания абстрактного описания исключены такие действия, как встрихивание доски, позволяющее передвинуть застрявшую фишку, или извлечение фишек с помощью ножа и повторное укладывание их в ящик. Исключено также описание правил игры, что позволяет обойтись без рассмотрения подробных сведений о физических манипуляциях.

Задача игры в восемь относится к семейству **задач со скользящими фишками**, которые часто используются в искусственном интеллекте для проверки новых алгоритмов поиска. Известно, что этот общий класс задач является NP-полным, поэтому вряд ли можно надеяться найти методы, которые в наихудшем случае были бы намного лучше по сравнению с алгоритмами поиска, описанными в этой и следующей главах. Задача игры в восемь имеет  $9! / 2 = 181\ 440$  достижимых состояний и легко решается. Задача игры в пятнадцать (на доске  $4 \times 4$ ) имеет около 1,3 триллиона состояний, и случайно выбранные ее экземпляры могут быть решены оптимальным образом за несколько миллисекунд с помощью наилучших алгоритмов поиска. Задача игры в двадцать четыре (на доске  $5 \times 5$ ) имеет количество состояний около  $10^{25}$ , и случайно выбранные ее экземпляры все еще весьма нелегко решить оптимальным образом с применением современных компьютеров и алгоритмов.

Цель **задачи с восемью ферзями** состоит в размещении восьми ферзей на шахматной доске таким образом, чтобы ни один ферзь не нападал на любого другого. (Ферзь атакует любую фигуру, находящуюся на одной и той же горизонтали, вертикали или диагонали.) На рис. 3.4 показана неудачная попытка поиска решения: ферзь, находящийся на самой правой вертикали, атакован ферзем, который находится вверху слева.

Несмотря на то что существуют эффективные специализированные алгоритмы решения этой задачи и всего семейства задач с  $n$  ферзями, она по-прежнему остается интересной экспериментальной задачей для алгоритмов поиска. Для нее применяются формулировки двух основных типов. В **инкрементной формулировке** предусматривается использование операторов, которые дополняют описание состояния, начиная с пустого состояния; для задачи с восемью ферзями это означает, что каждое действие приводит к добавлению к этому состоянию еще одного ферзя.



*Рис. 3.4. Почти готовое решение задачи с восемью ферзями (поиск окончательного решения оставляем читателю в качестве упражнения)*

❖ **Формулировка полного состояния** начинается с установки на доску всех восьми ферзей и предусматривает их дальнейшее перемещение. В том и другом случае стоимость пути не представляет интереса, поскольку важно лишь достигнуть конечного состояния. Первая инкрементная формулировка, которая может применяться при осуществлении попыток решения этой задачи, приведена ниже.

- **Состояния.** Состоянием является любое расположение ферзей на доске в количестве от 0 до 8.
- **Начальное состояние.** Отсутствие ферзей на доске.
- **Функция определения преемника.** Установка ферзя на любой пустой клетке.
- **Проверка цели.** На доске находится восемь ферзей, и ни один из них не атакован.

В этой формулировке требуется проверить  $64 \cdot 63 \cdot \dots \cdot 57 \approx 3 \times 10^{14}$  возможных последовательностей. В лучшей формулировке должно быть запрещено помещать ферзя на любую клетку, которая уже атакована, следующим образом.

- **Состояния.** Состояниями являются расположения с  $n$  ферзями ( $0 \leq n \leq 8$ ), по одному ферзю в каждой из находящихся слева  $n$  вертикалей, притом что ни один ферзь не нападает на другого.
- **Функция определения преемника.** Установка ферзя на любой клетке в находящейся слева пустой вертикали таким образом, чтобы он не был атакован каким-либо другим ферзем.

Эта формулировка позволяет сократить пространство состояний задачи с восемью ферзями с  $3 \times 10^{14}$  до 2057, и поиск решений значительно упрощается. С другой стороны, для 100 ферзей первоначальная формулировка определяет приблизительно  $10^{400}$  состояний, а улучшенная формулировка — около  $10^{52}$  состояний (см. упр. 3.5). Это — колоссальное сокращение, но улучшенное пространство состояний все еще слишком велико для того, чтобы с ним могли справиться алгоритмы, рассмат-

риваемые в данной главе. В главе 4 описана формулировка полного состояния, а в главе 5 приведен простой алгоритм, который позволяет легко решить задачу даже с миллионом ферзей.

## Реальные задачи

Выше уже было показано, как можно определить **задачу поиска маршрута** в терминах заданных местонахождений и переходов по связям между ними. Алгоритмы поиска маршрута используются в самых разных приложениях, таких как системы маршрутизации в компьютерных сетях, системы планирования военных операций и авиапутешествий. Обычно процесс определения таких задач является трудоемким. Рассмотрим упрощенный пример задачи планирования авиапутешествий, который задан следующим образом.

- **Состояния.** Каждое состояние представлено местонахождением (например, аэропортом) и текущим временем.
- **Начальное состояние.** Оно задается в условии задачи.
- **Функция определения преемника.** Эта функция возвращает состояния, которые следуют из выполнения любого полета, указанного в расписании (возможно, с дополнительным указанием класса и места), отправления позже по сравнению с текущим временем с учетом продолжительности переезда внутри самого аэропорта, а также из одного аэропорта в другой.
- **Проверка цели.** Находимся ли мы в месте назначения к некоторому заранее заданному времени?
- **Стоимость пути.** Зависит от стоимости билета, времени ожидания, продолжительности полета, таможенных и иммиграционных процедур, комфорта места, времени суток, типа самолета, скидок для постоянных пассажиров и т.д.

В коммерческих консультационных системах планирования путешествий используется формулировка задачи такого типа со многими дополнительными усложнениями, которые требуются для учета чрезвычайно запутанных структур определения платы за проезд, применяемых в авиакомпаниях. Но любой опытный путешественник знает, что не все авиапутешествия проходят согласно плану. Действительно качественная система должна предусматривать планы действий в непредвиденных ситуациях (такие как страховое резервирование билетов на альтернативные рейсы) в такой степени, которая соответствует стоимости и вероятности нарушения первоначального плана.

**Задачи планирования обхода** тесно связаны с задачами поиска маршрута, но с одним важным исключением. Рассмотрим, например, задачу: “Посетить каждый город, показанный на рис. 3.1, по меньшей мере один раз, начав и окончив путешествие в Бухаресте”. Как и при поиске маршрута, действия соответствуют поездкам из одного смежного города в другой. Но пространство состояний является совершенно другим. Каждое состояние должно включать не только текущее местонахождение, но также и множество городов, которые посетил агент. Поэтому первоначальным состоянием должно быть “В Бухаресте; посещен {Бухарест}”, а типичным промежуточным состоянием — “В Васлуй; посещены {Бухарест, Урзичени, Васлуй}”, тогда как проверка цели должна предусматривать определение того, находится ли агент в Бухаресте и посетил ли он все 20 городов.

Одной из задач планирования обхода является **задача коммивояжера** (Traveling Salesperson Problem — TSP), по условию которой каждый город должен быть посещен только один раз. Назначение ее состоит в том, чтобы найти самый короткий путь обхода. Как известно, эта задача является NP-трудной, но на улучшение возможностей алгоритмов TSP были затрачены колоссальные усилия. Кроме планирования поездок коммивояжеров, эти алгоритмы использовались для решения таких задач, как планирование перемещений автоматических сверл при отработке печатных плат и организация работы средств снабжения в производственных цехах.

Задача **компоновки СБИС** требует позиционирования миллионов компонентов и соединений на микросхеме для минимизации площади, схемных задержек, паразитных емкостей и максимизации выхода готовой продукции. Задача компоновки следует за этапом логического проектирования и обычно подразделяется на две части: **компоновка ячеек** и **маршрутизация каналов**. При компоновке ячеек простейшие компоненты схемы группируются по ячейкам, каждая из которых выполняет некоторую известную функцию. Каждая ячейка имеет постоянную форму (размеры и площадь) и требует создания определенного количества соединений с каждой из остальных ячеек. Требуется разместить ячейки на микросхеме таким образом, чтобы они не перекрывались и оставалось место для прокладки соединительных проводов между ячейками. При маршрутизации каналов происходит поиск конкретного маршрута для каждого проводника через зазоры между ячейками. Эти задачи поиска являются чрезвычайно сложными, но затраты на их решение, безусловно, оправдываются. В главе 4 приведены некоторые алгоритмы, позволяющие решать эти задачи.

**Задача управления навигацией робота** представляет собой обобщение описанной выше задачи поиска маршрута. В этой задаче вместо дискретного множества маршрутов рассматривается ситуация, в которой робот может перемещаться в непрерывном пространстве с бесконечным (в принципе) множеством возможных действий и состояний. Если требуется обеспечить циклическое перемещение робота по плоской поверхности, то пространство фактически может рассматриваться как двухмерное, а если робот оборудован верхними и нижними конечностями или колесами, которыми также необходимо управлять, то пространство поиска становится многомерным. Даже для того чтобы сделать это пространство поиска конечным, требуются весьма развитые методы. Некоторые из этих методов рассматриваются в главе 25. Изначальная сложность задачи усугубляется тем, что при управлении реальными роботами необходимо учитывать ошибки в показаниях датчиков, а также отклонения в работе двигательных средств управления.

Решение задачи **автоматического упорядочения сборки** сложных объектов роботом было впервые продемонстрировано на примере робота Freddy [1044]. С тех пор прогресс в этой области происходил медленно, но уверенно, и в настоящее время достигнуто такое положение, что стала экономически выгодной сборка таких неординарных объектов, как электродвигатели. В задачах сборки цель состоит в определении последовательности, в которой должны быть собраны детали некоторого объекта. Если выбрана неправильная последовательность, то в дальнейшем нельзя будет найти способ добавления некоторой детали к этой последовательности без отмены определенной части уже выполненной работы. Проверка возможности выполнения некоторого этапа в последовательности представляет собой сложную геометрическую задачу поиска, тесно связанную с задачей навигации робота. Поэтому одним из дорогостоящих этапов решения задачи упорядочения сборки является

формирование допустимых преемников. Любой практически применимый алгоритм должен предотвращать необходимость поиска во всем пространстве состояний, за исключением крошечной его части. Еще одной важной задачей сборки является **проектирование молекулы белка**, цель которой состоит в определении последовательности аминокислот, способных сложиться в трехмерный белок с нужными свойствами, предназначенный для лечения некоторых заболеваний.

В последние годы выросла потребность в создании программных роботов, которые осуществляют **поиск в Internet**, находя ответы на вопросы, отыскивая требуемую информацию или совершая торговые сделки. Это — хорошее приложение для методов поиска, поскольку Internet легко представить концептуально в виде графа, состоящего из узлов (страниц), соединенных с помощью ссылок. Полное описание задачи поиска в Internet отложим до главы 10.

### **3.3. ПОИСК РЕШЕНИЙ**

---

Сформулировав определенные задачи, необходимо найти их решение. Такая цель достигается посредством поиска в пространстве состояний. В настоящей главе рассматриваются методы поиска, в которых используются явно заданное **дерево поиска**, создаваемое с помощью начального состояния и функции определения преемника, которые совместно задают пространство состояний. Вообще говоря, вместо дерева поиска может применяться граф поиска, если одно и то же состояние может быть достигнуто с помощью многих путей. Это важное дополнение рассматривается в разделе 3.5.

На рис. 3.5 показаны некоторые расширения дерева поиска, предназначенного для определения маршрута от Арада до Бухареста. Корнем этого дерева поиска является **поисковый узел**, соответствующий начальному состоянию, *In(Arad)*. Первый этап состоит в проверке того, является ли это состояние целевым. Безусловно, что оно не является таковым, но необходимо предусмотреть соответствующую проверку, чтобы можно было решать задачи, содержащие в себе готовое решение, такие как “начав путешествие с города Арад, прибыть в город Арад”. А в данном случае текущее состояние не является целевым, поэтому необходимо рассмотреть некоторые другие состояния. Такой этап осуществляется путем **развертывания** текущего состояния, т.е применения функции определения преемника к текущему состоянию для **формирования** в результате этого нового множества состояний. В данном случае будут получены три новых состояния: *In(Sibiu)*, *In(Timisoara)* и *In(Zerind)*. Теперь необходимо определить, какой из этих трех вариантов следует рассматривать дальше.

В этом и состоит суть поиска — пока что проверить один вариант и отложить другие в сторону, на тот случай, что первый вариант не приведет к решению. Предположим, что вначале выбран город Сибиу. Проведем проверку для определения того, соответствует ли он целевому состоянию (не соответствует), а затем развернем узел *Sibiu* для получения состояний *In(Arad)*, *In(Fagaras)*, *In(Oradea)* и *In(RimnicuVilcea)*. После этого можно выбрать любое из этих четырех состояний либо вернуться и выбрать узел *Timisoara* или *Zerind*. Необходимо снова и снова выбирать, проверять и развертывать узлы до тех пор, пока не будет найдено решение или не останется больше состояний, которые можно было бы развернуть. Порядок, в котором происходит развертывание состояний, определяется **стратегией поиска**. Общий алгоритм поиска в дереве неформально представлен в листинге 3.2.

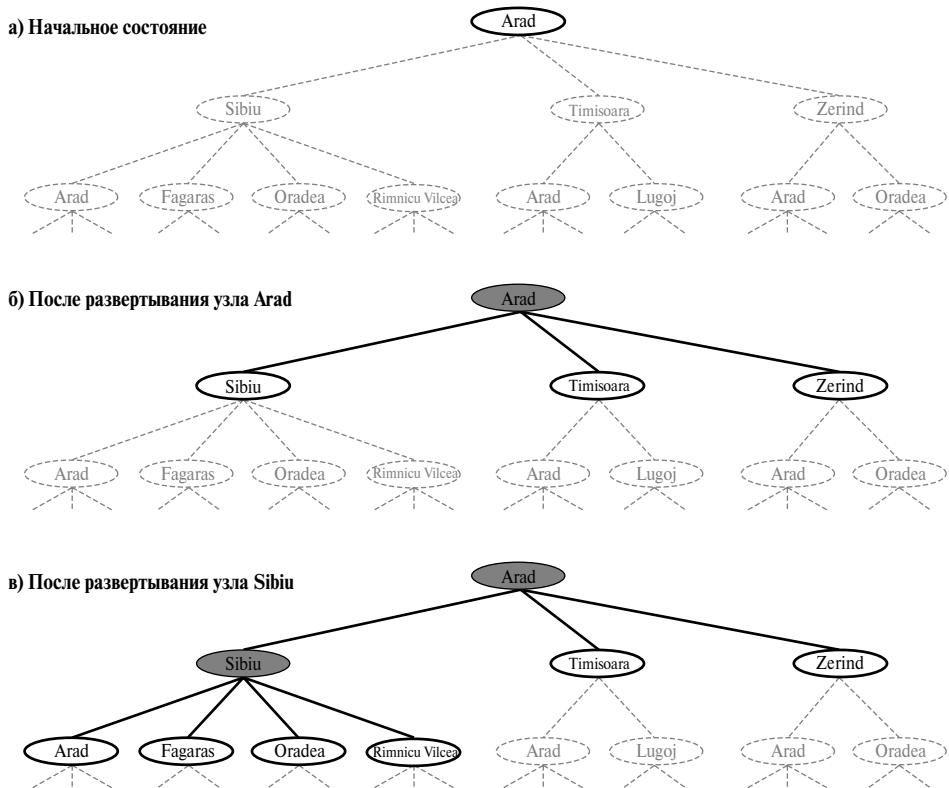


Рис. 3.5. Частично развернутые деревья поиска, предназначенные для определения маршрута от Арада до Бухареста. Развернутые узлы затенены; узлы, которые были сформированы, но еще не развернуты, выделены полужирным контуром; узлы, которые еще не были сформированы, обозначены тонкими штриховыми линиями

### Листинг 3.2. Неформальное описание общего алгоритма поиска в дереве

```

function Tree-Search(problem, strategy) returns решение solution
    или индикатор неудачи failure
    инициализировать дерево поиска с использованием начального
    состояния задачи problem
loop do
    if нет кандидатов на развертывание then return индикатор
        неудачи failure
    выбрать листовой узел для развертывания в соответствии
        со стратегией strategy
    if этот узел содержит целевое состояние
        then return соответствующее решение solution
    else развернуть этот узел и добавить полученные узлы
        к дереву поиска

```

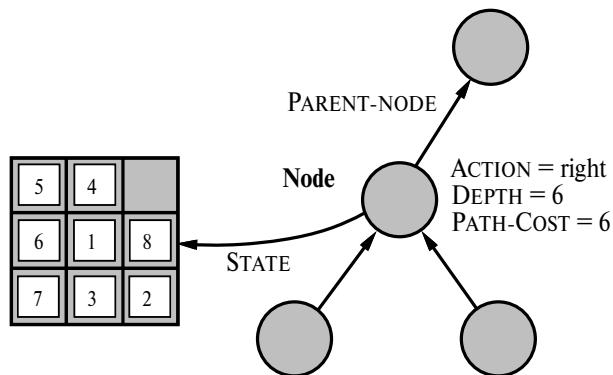
Необходимо учитывать различие между пространством состояний и деревом поиска. В пространстве состояний для задачи поиска маршрута имеется только 20 состояний, по одному для каждого города. Но количество путей в этом пространстве состояний является бесконечным, поэтому дерево поиска имеет бесконечное количество узлов. Напри-

мер, первыми тремя путями любой бесконечной последовательности путей являются маршруты Арад–Сибиу, Арад–Сибиу–Арад, Арад–Сибиу–Арад–Сибиу. (Безусловно, качественный алгоритм поиска должен исключать возможность формирования таких повторяющихся путей; в разделе 3.5 показано, как этого добиться.)

Существует множество способов представления узлов, но здесь предполагается, что узел представляет собой структуру данных с пятью компонентами, которые описаны ниже.

- **State.** Состояние в пространстве состояний, которому соответствует данный узел.
- **Parent-Node.** Узел в дереве поиска, применявшийся для формирования данного узла (родительский узел).
- **Action.** Действие, которое было применено к родительскому узлу для формирования данного узла.
- **Path-Cost.** Стоимость пути (от начального состояния до данного узла), показанного с помощью указателей родительских узлов, которую принято обозначать как  $g(n)$ .
- **Depth.** Количество этапов пути от начального состояния, называемое также *глубиной*.

Необходимо учитывать различие между узлами и состояниями. *Узел* — это учетная структура данных, применяемая для представления дерева поиска, а *состояние* соответствует конфигурации мира. Поэтому узлы лежат на конкретных путях, которые определены с помощью указателей Parent-Node, а состояния — нет. Кроме того, два разных узла могут включать одно и то же состояние мира, если это состояние формируется с помощью двух различных путей поиска. Структура данных узла показана на рис. 3.6.



*Рис. 3.6. Узлы представляют собой структуры данных, с помощью которых формируется дерево поиска. Каждый узел имеет родительский узел, содержит данные о состоянии и имеет различные вспомогательные поля. Стрелки направлены от дочернего узла к родительскому*

Необходимо также представить коллекцию узлов, которые были сформированы, но еще не развернуты; такая коллекция называется **периферией**. Каждый элемент

периферии представляет собой **листовой узел**, т.е. узел, не имеющий преемников в дереве. На рис. 3.5 периферия каждого дерева состоит из узлов с полужирными контурами. Простейшим представлением периферии может служить множество узлов. Тогда стратегия поиска должна быть выражена в виде функции, которая выбирает определенным образом из этого множества следующий узел, подлежащий развертыванию. Хотя данный подход концептуально является несложным, он может оказаться дорогостоящим с вычислительной точки зрения, поскольку функцию, предусмотренную в этой стратегии, возможно, придется применить к каждому элементу в указанном множестве для выбора наилучшего из них. Поэтому предполагается, что коллекция узлов реализована в виде **очереди**. Операции, применимые к любой очереди, состоят в следующем.

- **Make-Queue(*element*, ...)**. Создает очередь с заданным элементом (элементами).
- **Empty?(*queue*)**. Возвращает истинное значение, только если в очереди больше нет элементов.
- **First(*queue*)**. Возвращает первый элемент в очереди.
- **Remove-First(*queue*)**. Возвращает элемент **First(*queue*)** и удаляет его из очереди.
- **Insert(*element*, *queue*)**. Вставляет элемент в очередь и возвращает результатирующую очередь. (Ниже будет показано, что в очередях различных типов вставка элементов осуществляется в различном порядке.)
- **Insert-All(*elements*, *queue*)**. Вставляет множество элементов в очередь и возвращает результатирующую очередь.

С помощью этих определений мы можем записать более формальную версию общего алгоритма поиска в дереве, показанную в листинге 3.3.

**Листинг 3.3. Общий алгоритм поиска в дереве.** Следует учитывать, что фактический параметр **fringe** (периферия) должен представлять собой пустую очередь, а порядок поиска зависит от типа очереди. Функция **Solution** возвращает последовательность действий, полученную путем прохождения по указателям на родительские узлы в обратном направлении, к корню

---

```

function Tree-Search(problem, fringe) returns решение solution
    или индикатор неудачи failure

    fringe  $\leftarrow$  Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if Empty?(fringe) then return индикатор неудачи failure
        node  $\leftarrow$  Remove-First(fringe)
        if Goal-Test[problem] применительно к State[node]
            завершается успешно
            then return Solution(node)
        fringe  $\leftarrow$  Insert-All(Expand(node, problem), fringe)

function Expand(node, problem) returns множество узлов successors

    successors  $\leftarrow$  пустое множество
    for each <action, result> in Successor-Fn[problem](State[node]) do
        s  $\leftarrow$  новый узел

```

```

State[s] ← result
Parent-Node[s] ← node
Action[s] ← action
Path-Cost[s] ← Path-Cost[node] + Step-Cost(node, action, s)
Depth[s] ← Depth[node] + 1
добавить узел s к множеству successors
return successors

```

---

## Измерение производительности решения задачи

Результатом применения любого алгоритма решения задачи является либо неудачное завершение, либо получение решения. (Некоторые алгоритмы могут входить в бесконечный цикл и не возвращать никакого результата.) Мы будем оценивать производительность алгоритма с помощью четырех показателей, описанных ниже.

- **↗ Полнота.** Гарантирует ли алгоритм обнаружение решения, если оно имеется?
- **↗ Оптимальность.** Обеспечивает ли данная стратегия нахождение оптимального решения, в соответствии с определением, приведенным на с. 115?
- **↗ Временная сложность.** За какое время алгоритм находит решение?
- **↗ Пространственная сложность.** Какой объем памяти необходим для осуществления поиска?

Временная и пространственная сложность всегда анализируются с учетом определенного критерия измерения сложности задачи. В теоретической компьютерной науке типичным критерием является размер графа пространства состояний, поскольку этот граф рассматривается как явно заданная структура данных, которая является входной для программы поиска. (Примером этого может служить карта Румынии.) В искусственном интеллекте, где граф представлен неявно с помощью начального состояния и функции определения преемника и часто является бесконечным, сложность выражается в терминах трех величин:  $b$  — **↗ коэффициент ветвления** или максимальное количество преемников любого узла,  $d$  — глубина самого поверхностного целевого узла и  $m$  — максимальная длина любого пути в пространстве состояний.

Временная сложность часто измеряется в терминах количества узлов, вырабатываемых<sup>5</sup> в процессе поиска, а пространственная сложность — в терминах максимального количества узлов, хранимых в памяти.

Чтобы оценить эффективность любого алгоритма поиска, можно рассматривать только **↗ стоимость поиска**, которая обычно зависит от временной сложности, но может также включать выражение для оценки использования памяти, или применять **↗ суммарную стоимость**, в которой объединяется стоимость поиска и стоимость пути найденного решения. Для задачи поиска маршрута от Арада до Бухареста стоимость поиска представляет собой количество времени, затраченного на этот поиск, а стоимость решения выражает общую длину пути в километрах. Поэтому для вы-

<sup>5</sup> В некоторых работах вместо этого для измерения временной сложности применяется количество операций развертывания узлов. Эти два критерия различаются не больше, чем на коэффициент  $b$ . С точки зрения авторов, время выполнения операции развертывания узла растет пропорционально количеству узлов, вырабатываемых при этом развертывании.

числения суммарной стоимости нам придется складывать километры и миллисекунды. Между этими двумя единицами измерения не определен “официальный курс обмена”, но в данном случае было бы резонно преобразовывать километры в миллисекунды с использованием оценки средней скорости автомобиля (поскольку для данного агента важным является именно время). Это позволяет рассматриваемому агенту найти оптимальную точку компромисса, в которой дальнейшие вычисления для поиска более короткого пути становятся непродуктивными. Описание более общей задачи поиска компромисса между различными ценностями будет продолжено в главе 16.

### 3.4. СТРАТЕГИИ НЕИНФОРМИРОВАННОГО ПОИСКА

В данном разделе рассматриваются пять стратегий поиска, которые известны под названием **неинформированного поиска** (называемого также **слепым поиском**). Этот термин означает, что в данных стратегиях не используется дополнительная информация о состояниях, кроме той, которая представлена в определении задачи. Все, на что они способны, — вырабатывать преемников и отличать целевое состояние от нецелевого. Стратегии, позволяющие определить, является ли одно нецелевое состояние “более многообещающим” по сравнению с другим, называются стратегиями **информированного поиска**, или **эвристического поиска**; они рассматриваются в главе 4. Все стратегии поиска различаются тем, в каком порядке происходит развертывание узлов.

#### Поиск в ширину

**Поиск в ширину** — это простая стратегия, в которой вначале развертывается корневой узел, затем — все преемники корневого узла, после этого развертываются преемники этих преемников и т.д. Вообще говоря, при поиске в ширину, прежде чем происходит развертывание каких-либо узлов на следующем уровне, развертываются все узлы на данной конкретной глубине в дереве поиска.

Поиск в ширину может быть реализован путем вызова процедуры `Tree-Search` с пустой периферией, которая представляет собой последовательную очередь (`First-In-First-Out` — FIFO), гарантирующую, что прежде всего будут развернуты узлы, которые должны посещаться первыми. Иными словами, к организации поиска в глубину приводит вызов процедуры `Tree-Search(problem, FIFO-Queue())`. В очереди FIFO предусмотрена вставка всех вновь сформированных преемников в конец очереди, а это означает, что поверхностные узлы развертываются прежде, чем более глубокие. На рис. 3.7 показан ход поиска в простом бинарном дереве.

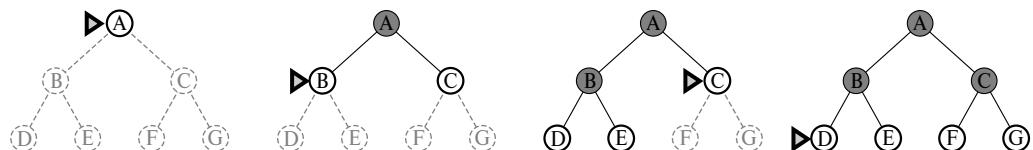


Рис. 3.7. Поиск в ширину в простом бинарном дереве. На каждом этапе узел, подлежащий развертыванию в следующую очередь, обозначается маркером ►

Проведем оценку поиска в ширину с использованием четырех критериев, описанных в предыдущем разделе. Вполне очевидно, что этот поиск является полным — если самый поверхностный целевой узел находится на некоторой конечной глубине  $d$ , то поиск в ширину в конечном итоге позволяет его обнаружить после развертывания всех более поверхностных узлов (при условии, что коэффициент ветвления  $b$  является конечным). Самый поверхностный целевой узел не обязательно является оптимальным; формально поиск в ширину будет оптимальным, если стоимость пути выражается в виде неубывающей функции глубины узла. (Например, такое предположение оправдывается, если все действия имеют одинаковую стоимость.)

До сих пор приведенное выше описание поиска в ширину не предвещало никаких неприятностей. Но такая стратегия не всегда является оптимальной; чтобы понять, с чем это связано, необходимо определить, какое количество времени и какой объем памяти требуются для выполнения поиска. Для этого рассмотрим гипотетическое пространство состояний, в котором каждое состояние имеет  $b$  преемников. Корень этого дерева поиска вырабатывает  $b$  узлов на первом уровне, каждый из которых вырабатывает еще  $b$  узлов, что соответствует общему количеству узлов на втором уровне, равному  $b^2$ . Каждый из них также вырабатывает  $b$  узлов, что приводит к получению  $b^3$  узлов на третьем уровне, и т.д. А теперь предположим, что решение находится на глубине  $d$ . В наихудшем случае на уровне  $d$  необходимо развернуть все узлы, кроме последнего (поскольку сам целевой узел не развертывается), что приводит к выработке  $b^{d+1}-b$  узлов на уровне  $d+1$ . Это означает, что общее количество выработанных узлов равно:

$$b + b^2 + b^3 + \dots + b^d + (b^{d+1}-b) = O(b^{d+1})$$

Каждый выработанный узел должен оставаться в памяти, поскольку он либо относится к периферии, либо является предком периферийного узла. Итак, пространственная сложность становится такой же, как и временная (с учетом добавления одного узла, соответствующего корню).

Поэтому исследователи, выполняющие анализ сложности алгоритма, огорчаются (или восхищаются, если им нравится преодолевать трудности), столкнувшись с экспоненциальными оценками сложности, такими как  $O(b^{d+1})$ . В табл. 3.1 показано, с чем это связано. В ней приведены требования ко времени и к объему памяти при поиске в ширину с коэффициентом ветвления  $b=10$  для различных значений глубины решения  $d$ . При составлении этой таблицы предполагалось, что в секунду может быть сформировано 10 000 узлов, а для каждого узла требуется 1000 байтов памяти. Этим предположением приблизительно соответствуют многие задачи поиска при их решении на любом современном персональном компьютере (с учетом повышающего или понижающего коэффициента 100).

На основании табл. 3.1 можно сделать два важных вывода. Прежде всего, *при поиске в ширину наиболее сложной проблемой по сравнению со значительным временем выполнения является обеспечение потребностей в памяти*. Затраты времени, равные 31 часу, не кажутся слишком значительными при ожидании решения важной задачи с глубиной 8, но лишь немногие компьютеры имеют терабайт оперативной памяти, который для этого требуется. К счастью, существуют другие стратегии поиска, которые требуют меньше памяти.

Второй вывод состоит в том, что требования ко времени все еще остаются важным фактором. Если рассматриваемая задача имеет решение на глубине 12, то (с учетом при-

нятых предположений) потребуется 35 лет на поиск в ширину (а в действительности на любой неинформированный поиск), чтобы найти ее решение. Вообще говоря, *задачи поиска с экспоненциальной сложностью невозможно решить с помощью неинформированных методов во всех экземплярах этих задач, кроме самых небольших.*

**Таблица 3.1. Потребности во времени и объеме памяти для поиска в ширину. Приведенные здесь данные получены при следующих предположениях: коэффициент ветвления —  $b=10$ ; скорость формирования — 10 000 узлов/секунда; объем памяти — 1000 байтов/узел**

Глубина	Количество узлов	Время	Память
2	1100	0,11 секунды	1 мегабайт
4	111 100	11 секунд	106 мегабайтов
6	$10^7$	19 минут	10 гигабайтов
8	$10^9$	31 час	1 терабайт
10	$10^{11}$	129 суток	101 терабайт
12	$10^{13}$	35 лет	10 петабайтов
14	$10^{15}$	3523 года	1 эксабайт

### Поиск по критерию стоимости

Поиск в ширину является оптимальным, если стоимости всех этапов равны, поскольку в нем всегда развертывается самый поверхностный неразвернутый узел. С помощью простого дополнения можно создать алгоритм, который является оптимальным при любой функции определения стоимости этапа. Вместо развертывания самого поверхностного узла **поиск по критерию стоимости** обеспечивает развертывание узла  $n$  с наименьшей стоимостью пути. Обратите внимание на то, что, если стоимости всех этапов равны, такой поиск идентичен поиску в ширину.

При поиске по критерию стоимости учитывается не количество этапов, имеющихся в пути, а только их суммарная стоимость. Поэтому процедура этого поиска может войти в бесконечный цикл, если окажется, что в ней развернут узел, имеющий действие с нулевой стоимостью, которое снова указывает на то же состояние (например, действие *NoOp*). Можно гарантировать полноту поиска при условии, что стоимость каждого этапа больше или равна некоторой небольшой положительной константе  $\epsilon$ . Это условие является также достаточным для обеспечения оптимальности. Оно означает, что стоимость пути всегда возрастает по мере прохождения по этому пути. Из данного свойства легко определить, что такой алгоритм развертывает узлы в порядке возрастания стоимости пути. Поэтому первый целевой узел, выбранный для развертывания, представляет собой оптимальное решение. (Напомним, что в процедуре *Tree-Search* проверка цели применяется только к тем узлам, которые выбраны для развертывания.) Рекомендуем читателю попытаться воспользоваться этим алгоритмом для поиска кратчайшего пути до Бухареста.

Поиск по критерию стоимости направляется с учетом стоимостей путей, а не значений глубины в дереве поиска, поэтому его сложность не может быть легко охарактеризована в терминах  $b$  и  $d$ . Вместо этого предположим, что  $C^*$  — стоимость оптимального решения, и допустим, что стоимость каждого действия составляет, по меньшей мере,  $\epsilon$ . Это означает, что времененная и пространственная сложность этого алгоритма в наихудшем случае составляет  $O(b^{1+C^*/\epsilon})$ , т.е. может быть намного больше, чем  $b^d$ . Это связано с тем, что процедуры поиска по критерию стоимости

могут и часто выполняют проверку больших деревьев, состоящих из мелких этапов, прежде чем перейти к исследованию путей, в которые входят крупные, но, возможно, более полезные этапы. Безусловно, если все стоимости этапов равны, то выражение  $b^{1+\lfloor c^*/\epsilon \rfloor}$  равняется  $b^d$ .

### Поиск в глубину

При **поиске в глубину** всегда развертывается самый глубокий узел в текущей периферии дерева поиска. Ход такого поиска показан на рис. 3.8. Поиск непосредственно переходит на самый глубокий уровень дерева поиска, на котором узлы не имеют преемников. По мере того как эти узлы развертываются, они удаляются из периферии, поэтому в дальнейшем поиск “возобновляется” со следующего самого поверхностного узла, который все еще имеет неисследованных преемников.

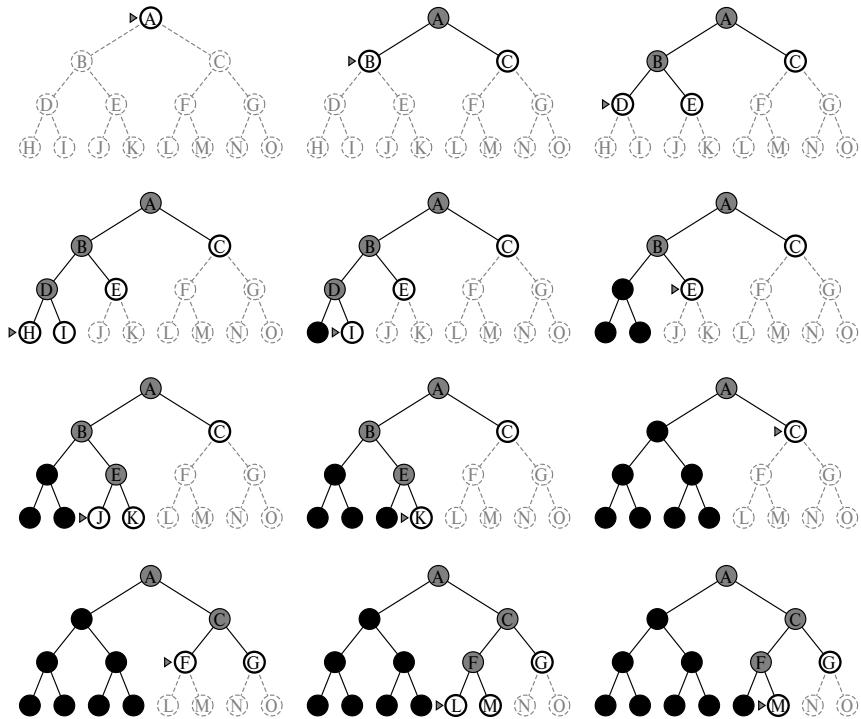


Рис. 3.8. Поиск в глубину в бинарном дереве. Узлы, которые были развернуты и не имеют потомков в этой периферии, могут быть удалены из памяти; эти узлы обозначены черным цветом. Предполагается, что узлы на глубине 3 не имеют преемников и единственным целевым узлом является M

Эта стратегия может быть реализована в процедуре Tree-Search с помощью очереди LIFO (Last-In-First-Out), называемой также *стеком*. В качестве альтернативы способу реализации на основе процедуры Tree-Search поиск в глубину часто реализуют с помощью рекурсивной функции, вызывающей саму себя в каждом из дочерних узлов по очереди. (Рекурсивный алгоритм поиска в глубину, в котором предусмотрен предел глубины, приведен в листинге 3.4.)

Поиск в глубину имеет очень скромные потребности в памяти. Он требует хранения только единственного пути от корня до листового узла, наряду с оставшимися неразвернутыми сестринскими узлами для каждого узла пути. После того как был развернут некоторый узел, он может быть удален из памяти, коль скоро будут полностью исследованы все его потомки (см. рис. 3.8). Для пространства состояний с коэффициентом ветвления  $b$  и максимальной глубиной  $m$  поиск в глубину требует хранения только  $bm+1$  узлов. Используя такие же предположения, как и в табл. 3.1, и допуская, что узлы, находящиеся на той же глубине, что и целевой узел, не имеют преемников, авторы определили, что на глубине  $d=12$  для поиска в глубину требуется 118 килобайтов вместо 10 петабайтов, т. е. потребность в пространстве уменьшается примерно в 10 миллиардов раз.

В одном из вариантов поиска в глубину, называемом **поиском с возвратами**, используется еще меньше памяти. При поиске с возвратами каждый раз формируется только один преемник, а не все преемники; в каждом частично развернутом узле запоминается информация о том, какой преемник должен быть сформирован следующим. Таким образом, требуется только  $O(m)$  памяти, а не  $O(bm)$ . При поиске с возвратами применяется еще один прием, позволяющий экономить память (и время); идея его состоит в том, чтобы при формировании преемника должно непосредственно модифицироваться описание текущего состояния, а не осуществляться его предварительное копирование. При этом потребность в памяти сокращается до объема, необходимого для хранения только одного описания состояния и  $O(m)$  действий. Но для успешного применения данного приема нужно иметь возможность отменять каждую модификацию при возврате, выполняемом для формирования следующего преемника. При решении задач с объемными описаниями состояния, таких как роботизированная сборка, применение указанных методов модификации состояний становится важнейшим фактором успеха.

Недостатком поиска в глубину является то, что в нем может быть сделан неправильный выбор и переход в тупиковую ситуацию, связанную с прохождением вниз по очень длинному (или даже бесконечному) пути, притом что другой вариант мог бы привести к решению, находящемуся недалеко от корня дерева поиска. Например, на рис. 3.8 поиск в глубину потребовал бы исследования всего левого поддерева, даже если бы целевым узлом был узел  $C$ , находящийся в правом поддереве. А если бы целевым узлом был также узел  $J$ , менее приемлемый по сравнению с узлом  $C$ , то поиск в глубину возвратил бы в качестве решения именно его; это означает, что поиск в глубину не является оптимальным. Кроме того, если бы левое поддерево имело неограниченную глубину, но не содержало бы решений, то поиск в глубину так никогда бы и не закончился; это означает, что данный алгоритм — не полный. В наихудшем случае поиск в глубину формирует все  $O(b^m)$  узлов в дереве поиска, где  $m$  — максимальная глубина любого узла. Следует отметить, что  $m$  может оказаться гораздо больше по сравнению с  $d$  (глубиной самого поверхностного решения) и является бесконечным, если дерево имеет неограниченную глубину.

## Поиск с ограничением глубины

Проблему неограниченных деревьев можно решить, предусматривая применение во время поиска в глубину заранее определенного предела глубины  $\ell$ . Это означает, что узлы на глубине  $\ell$  рассматриваются таким образом, как если бы они не имели

преемников. Такой подход называется **поиском с ограничением глубины**. Применение предела глубины позволяет решить проблему бесконечного пути. К сожалению, в этом подходе также вводится дополнительный источник неполноты, если будет выбрано значение  $\ell < d$ , иными словами, если самая поверхностная цель выходит за пределы глубины. (Такая ситуация вполне вероятна, если значение  $d$  неизвестно.) Кроме того, поиск с ограничением глубины будет неоптимальным при выборе значения  $\ell > d$ . Его времененная сложность равна  $O(b^\ell)$ , а пространственная сложность —  $O(b\ell)$ . Поиск в глубину может рассматриваться как частный случай поиска с ограничением глубины, при котором  $\ell = \infty$ .

Иногда выбор пределов глубины может быть основан на лучшем понимании задачи. Например, допустим, что на рассматриваемой карте Румынии имеется 20 городов. Поэтому известно, что если решение существует, то должно иметь длину не больше 19; это означает, что одним из возможных вариантов является  $\ell = 19$ . Но в действительности при внимательном изучении этой карты можно обнаружить, что любой город может быть достигнут из любого другого города не больше чем за 9 этапов. Это число, известное как **диаметр** пространства состояний, предоставляет нам лучший предел глубины, который ведет к более эффективному поиску с ограничением глубины. Но в большинстве задач приемлемый предел глубины остается неизвестным до тех пор, пока не будет решена сама задача.

Поиск с ограничением глубины может быть реализован как простая модификация общего алгоритма поиска в дереве или рекурсивного алгоритма поиска в глубину. Псевдокод реализации рекурсивного поиска с ограничением глубины приведен в листинге 3.4. Обратите внимание на то, что поиск с ограничением глубины может приводить к неудачным завершениям двух типов: стандартное значение *failure* указывает на отсутствие решения, а значение *cutoff* свидетельствует о том, что на заданном пределе глубины решения нет.

#### Листинг 3.4. Рекурсивная реализация поиска с ограничением глубины

```
function Depth-Limited-Search(problem, limit) returns решение result
    или индикатор неудачи failure/cutoff
    return Recursive-DLS(Make-Node(Initial-State[problem]),
                           problem, limit)

function Recursive-DLS(node, problem, limit) returns решение result
    или индикатор неудачи failure/cutoff
    cutoff_occurred?  $\leftarrow$  ложное значение
    if Goal-Test[problem](State[node]) then return Solution(node)
    else if Depth[node] = limit then return индикатор неудачи cutoff
    else for each преемник successor in Expand(node, problem) do
        result  $\leftarrow$  Recursive-DLS(successor, problem, limit)
        if result = cutoff then cutoff_occurred?  $\leftarrow$  истинное значение
        else if result  $\neq$  failure then return решение result
    if cutoff_occurred?
        then return индикатор неудачи cutoff
    else return индикатор неудачи failure
```

## Поиск в глубину с итеративным углублением

❖ **Поиск с итеративным углублением** (или, точнее, поиск в глубину с итеративным углублением) представляет собой общую стратегию, часто применяемую в сочетании с поиском в глубину, которая позволяет найти наилучший предел глубины. Это достигается путем постепенного увеличения предела (который вначале становится равным 0, затем 1, затем 2 и т.д.) до тех пор, пока не будет найдена цель. Такое событие происходит после того, как предел глубины достигает значения  $d$ , глубины самого поверхностного целевого узла. Данный алгоритм приведен в листинге 3.5. В поиске с итеративным углублением сочетаются преимущества поиска в глубину и поиска в ширину. Как и поиск в глубину, он характеризуется очень скромными требованиями к памяти, а именно, значением  $O(bd)$ . Как и поиск в ширину, он является полным, если коэффициент ветвления конечен, и оптимальным, если стоимость пути представляет собой неубывающую функцию глубины узла. На рис. 3.9 показаны четыре итерации применения процедуры Iterative-Deepening-Search к бинарному дереву поиска, где решение найдено в четвертой итерации.

**Листинг 3.5.** Алгоритм поиска с итеративным углублением, в котором повторно применяется поиск с ограничением глубины при последовательном увеличении пределов. Он завершает свою работу после того, как обнаруживается решение, или процедура поиска с ограничением глубины возвращает значение `failure`, а это означает, что решение не существует

---

```

function Iterative-Deepening-Search(problem) returns решение result
    или индикатор неудачи failure
    inputs: problem, задача

    for depth  $\leftarrow 0$  to  $\infty$  do
        result  $\leftarrow$  Depth-Limited-Search(problem, depth)
        if result  $\neq$  cutoff then return решение result

```

---

Поиск с итеративным углублением может на первый взгляд показаться расточительным, поскольку одни и те же состояния формируются несколько раз. Но, как оказалось, такие повторные операции не являются слишком дорогостоящими. Причина этого состоит в том, что в дереве поиска с одним и тем же (или почти одним и тем же) коэффициентом ветвления на каждом уровне большинство узлов находится на нижнем уровне, поэтому не имеет большого значения то, что узлы на верхних уровнях формируются многоократно. В поиске с итеративным углублением узлы на нижнем уровне (с глубиной  $d$ ) формируются один раз, те узлы, которые находятся на уровне, предшествующем нижнему, формируются дважды, и т.д., вплоть до дочерних узлов корневого узла, которые формируются  $d$  раз. Поэтому общее количество формируемых узлов выражается следующей формулой:

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \dots + (1)b^d$$

которая соответствует временной сложности порядка  $O(b^d)$ . Это количество можно сравнить с количеством узлов, формируемых при поиске в ширину:

$$N(\text{BFS}) = b + b^2 + \dots + b^d + (b^{d+1}-b)$$

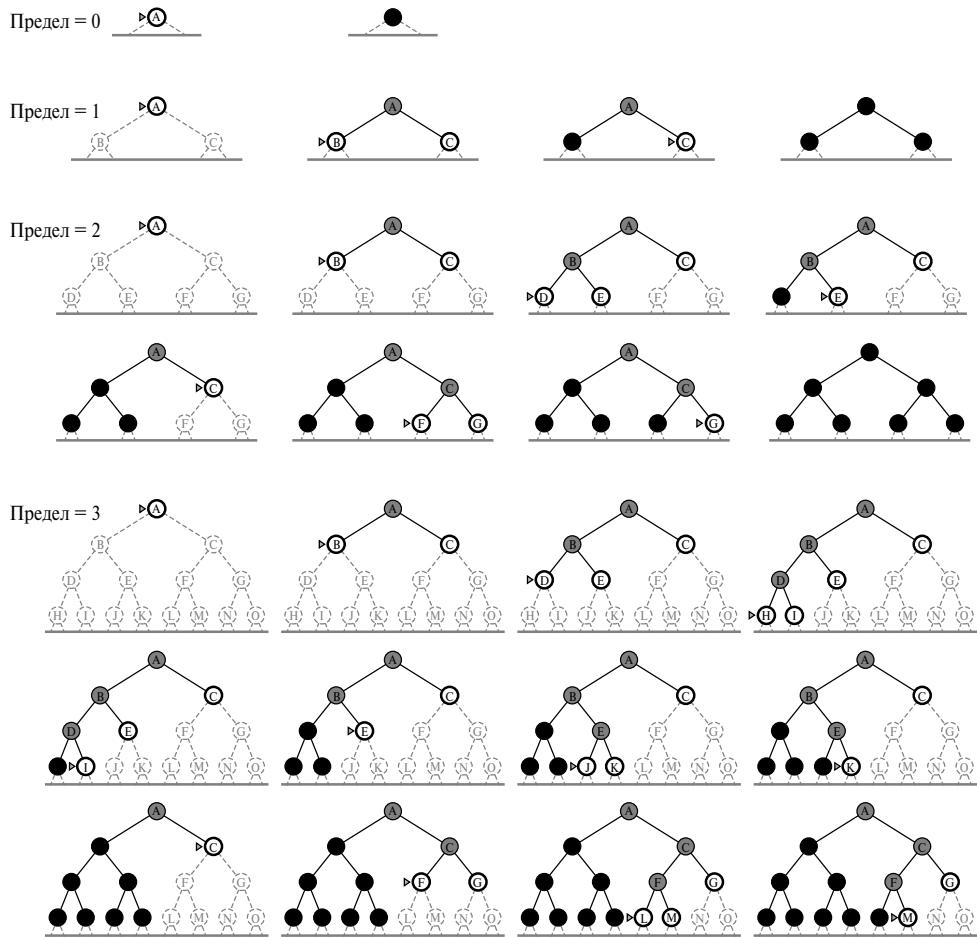


Рис. 3.9. Четыре итерации поиска с итеративным углублением в бинарном дереве

Следует отметить, что при поиске в ширину некоторые узлы формируются на глубине  $d+1$ , а при итеративном углублении этого не происходит. Результатом является то, что поиск с итеративным углублением фактически осуществляется быстрее, чем поиск в ширину, несмотря на повторное формирование состояний. Например, если  $b=10$  и  $d=5$ , то соответствующие оценки количества узлов принимают следующие значения:

$$\begin{aligned}N(\text{IDS}) &= 50 + 400 + 3000 + 20\ 000 + 100\ 000 = 123\ 450 \\N(\text{BFS}) &= 10 + 100 + 1000 + 10\ 000 + 100\ 000 + 999\ 990 = 1\ 111\ 100\end{aligned}$$

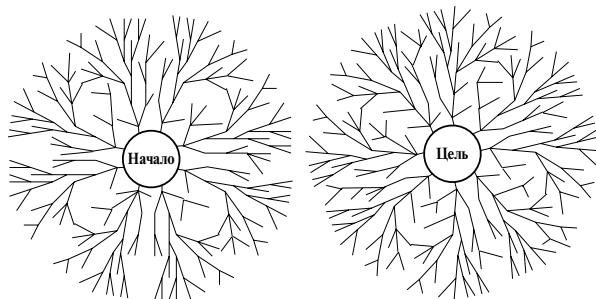
Вообще говоря, итеративное углубление — это предпочтительный метод неинформированного поиска при тех условиях, когда имеется большое пространство поиска, а глубина решения неизвестна.

Поиск с итеративным углублением аналогичен поиску в ширину в том отношении, что в нем при каждой итерации перед переходом на следующий уровень исследуется полный уровень новых узлов. На первый взгляд может показаться целесообразным разработка итеративного аналога поиска по критерию стоимости, который

унаследовал бы от последнего алгоритма гарантии оптимальности, позволяя вместе с тем исключить его высокие требования к памяти. Идея состоит в том, чтобы вместо увеличивающихся пределов глубины использовались увеличивающиеся пределы стоимости пути. Результатирующий алгоритм, получивший название **поиска с итеративным удлинением**, рассматривается в упр. 3.11. Но, к сожалению, было установлено, что поиск с итеративным удлинением характеризуется более существенными издержками, чем поиск по критерию стоимости.

### Двунаправленный поиск

В основе двунаправленного поиска лежит такая идея, что можно одновременно проводить два поиска (в прямом направлении, от начального состояния, и в обратном направлении, от цели), останавливаясь после того, как два процесса поиска встретятся на середине (рис. 3.10). Дело в том, что значение  $b^{d/2} + b^{d/2}$  гораздо меньше, чем  $b^d$ , или, как показано на этом рисунке, площадь двух небольших кругов меньше площади одного большого круга с центром в начале поиска, который охватывает цель поиска.



*Рис. 3.10. Схематическое представление двунаправленного поиска в том состоянии, когда он должен вскоре завершиться успешно после того, когда одна из ветвей, исходящих из начального узла, встретится с ветвью из целевого узла*

Двунаправленный поиск реализуется с помощью метода, в котором предусматривается проверка в одном или в обоих процессах поиска каждого узла перед его развертыванием для определения того, не находится ли он на периферии другого дерева поиска; в случае положительного результата проверки решение найдено. Например, если задача имеет решение на глубине  $d=6$  и в каждом направлении осуществляется поиск в ширину с последовательным развертыванием по одному узлу, то в самом худшем случае эти два процесса поиска встретятся, если в каждом из них будут развернуты все узлы на глубине 3, кроме одного. Это означает, что при  $b=10$  будет сформировано общее количество узлов, равное 22 200, а не 11 111 100, как при стандартном поиске в ширину. Проверка принадлежности узла к другому дереву поиска может быть выполнена за постоянное время с помощью хэш-таблицы, поэтому времененная сложность двунаправленного поиска определяется как  $O(b^{d/2})$ . В памяти необходимо хранить по крайней мере одно из деревьев поиска, для того, чтобы можно было выполнить проверку принадлежности к другому дереву, поэтому пространственная сложность также определяется как  $O(b^{d/2})$ . Такие требования к пространству являются одним из наиболее существенных недостатков двунаправленного поиска. Этот алгоритм является полным и оптимальным (при единообразных стоимостях эта-

пов), если оба процесса поиска осуществляются в ширину; другие сочетания методов могут характеризоваться отсутствием полноты, оптимальности или того и другого.

Благодаря такому уменьшению временной сложности двунаправленный поиск становится привлекательным, но как организовать поиск в обратном направлении? Это не так легко, как кажется на первый взгляд. Допустим, что  $\preceq$  **предшественниками** узла  $n$ , определяемыми с помощью функции  $Pred(n)$ , являются все те узлы, для которых  $n$  служит преемником. Для двунаправленного поиска требуется, чтобы функция определения предшественника  $Pred(n)$  была эффективно вычислимой. Простейшим является такой случай, когда все действия в пространстве состояний обратимы таким образом, что  $Pred(n) = Succ^{-1}(n)$ , а другие случаи могут потребовать проявить значительную изобретательность.

Рассмотрим вопрос о том, что подразумевается под понятием “цель” при поиске “в обратном направлении от цели”. В задачах игры в восемь и поиска маршрута в Румынии имеется только одно целевое состояние, поэтому обратный поиск весьма напоминает прямой поиск. Если же имеется несколько явно перечисленных целевых состояний (например, два показанных на рис. 3.2 целевых состояния, в которых квадраты пола не содержат мусора), то может быть создано новое фиктивное целевое состояние, непосредственными предшественниками которого являются все фактические целевые состояния. Иным образом, формирования некоторых избыточных узлов можно избежать, рассматривая множество целевых состояний как единственное целевое состояние, каждым из предшественников которого также является множество состояний, а именно, множество состояний, имеющее соответствующего преемника в множестве целевых состояний (см. также раздел 3.6).

Наиболее сложным случаем для двунаправленного поиска является такая задача, в которой для проверки цели дано только неявное описание некоторого (возможно, большого) множества целевых состояний, например всех состояний, соответствующих проверке цели “мат” в шахматах. При обратном поиске потребовалось бы создать компактные описания “всех состояний, которые позволяют поставить мат с помощью хода  $m_1$ ”, и т.д.; эти описания нужно было бы сверять с состояниями, формируемыми при прямом поиске. Общего способа эффективного решения такой проблемы не существует.

### **Сравнение стратегий неинформированного поиска**

В табл. 3.2 приведено сравнение стратегий поиска в терминах четырех критериев оценки, сформулированных в разделе 3.4.

---

## **3.5. ПРЕДОТВРАЩЕНИЕ ФОРМИРОВАНИЯ ПОВТОРЯЮЩИХСЯ СОСТОЯНИЙ**

---

Вплоть до этого момента мы рассматривали все аспекты поиска, но игнорировали одно из наиболее важных усложнений в процессе поиска — вероятность появления непроизводительных затрат времени при развертывании состояний, которые уже встречались и были развернуты перед этим. При решении некоторых задач такая ситуация никогда не возникает; в них пространство состояний представляет собой дерево и поэтому имеется только один путь к каждому состоянию. В частности, эф-

фективной является формулировка задачи с восемью ферзями (в которой каждый новый ферзь помещается на самый левый пустой вертикальный ряд), и ее эффективность в значительной степени обусловлена именно этим — каждое состояние может быть достигнуто только по одному пути. А если бы задача с восемью ферзями была сформулирована таким образом, что любого ферзя разрешалось бы ставить на любую вертикаль, то каждого состояния с  $n$  ферзями можно было бы достичь с помощью  $n!$  различных путей.

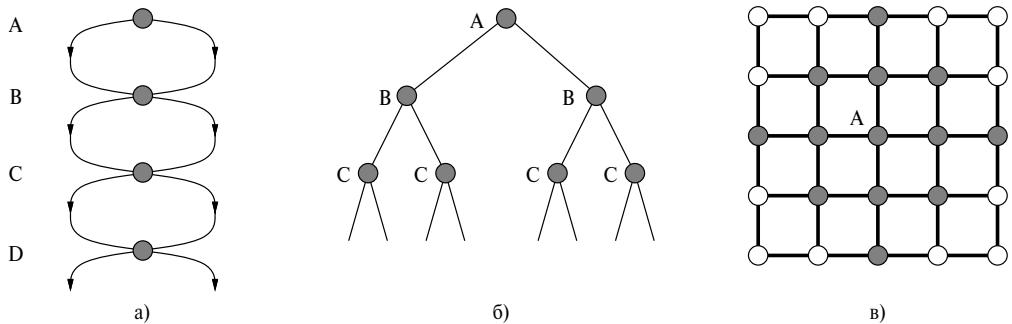
**Таблица 3.2. Оценка стратегий поиска**, где  $b$  — коэффициент ветвления;  $d$  — глубина самого поверхностного решения;  $m$  — максимальная глубина дерева поиска;  $\ell$  — предел глубины. Предостережения, обозначенные строчными буквами, означают следующее: <sup>a</sup> — полный, если коэффициент ветвления  $b$  конечен; <sup>b</sup> — полный, если стоимость каждого этапа  $\geq \epsilon$  при некотором положительном значении  $\epsilon$ ; <sup>c</sup> — оптимальный, если стоимости всех этапов являются одинаковыми; <sup>x</sup> — применимый, если в обоих направлениях осуществляется поиск в ширину

Характеристика	Поиск в ширину	Поиск по критерию стоимости	Поиск в глубину	Поиск с ограничением глубины	Поиск с итеративным углублением	Двунаправленный поиск (если он применим)
Полнота	Да <sup>a</sup>	Да <sup>a, b</sup>	Нет	Нет	Да <sup>a</sup>	Да <sup>a, x</sup>
Временная сложность	$O(b^{d+1})$	$O(b^{1+\lfloor c^*/\epsilon \rfloor})$	$O(B^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Пространственная сложность	$O(b^{d+1})$	$O(b^{1+\lfloor c^*/\epsilon \rfloor})$	$O(Bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Оптимальность	Да <sup>b</sup>	Да	Нет	Нет	Да <sup>b</sup>	Да <sup>b, x</sup>

В некоторых задачах повторяющиеся состояния являются неизбежными. К ним относятся все задачи, в которых действия являются обратимыми, такие как задачи поиска маршрута и игры со скользящими фишками. Деревья поиска для этих проблем бесконечны, но если мы отсечем некоторые из повторяющихся состояний, то сможем уменьшить дерево поиска до конечного размера, формируя только ту часть дерева, которая охватывает весь граф пространства состояний. Рассматривая только часть дерева поиска вплоть до некоторой постоянной глубины, можно легко обнаружить ситуации, в которых удаление повторяющихся состояний позволяет достичь экспоненциального уменьшения стоимости поиска. В крайнем случае пространство состояний размером  $d+1$  (рис. 3.11, а) становится деревом с  $2^d$  листьями (рис. 3.11, б). Более реалистичным примером может служить ~~прямоугольная решетка~~ **прямоугольная решетка**, как показано на рис. 3.11, в. В решетке каждое состояние имеет четырех преемников, поэтому дерево поиска, включающее повторяющиеся состояния, имеет  $4^d$  листьев, но существует приблизительно только  $2d^2$  различных состояний с  $d$  этапами достижения любого конкретного состояния. Для  $d=20$  это означает, что существует около триллиона узлов, но лишь примерно 800 различных состояний.

Таким образом, неспособность алгоритма обнаруживать повторяющиеся состояния может послужить причиной того, что разрешимая задача станет неразрешимой. Такое обнаружение обычно сводится к тому, что узел, подлежащий развертыванию, сравнивается с теми узлами, которые уже были развернуты; если обнаружено совпадение, то алгоритм распознал наличие двух путей в одно и то же состояние и может отбросить один из них.

При поиске в глубину в памяти хранятся только те узлы, которые лежат на пути от корня до текущего узла. Сравнение этих узлов с текущим узлом позволяет алгоритму обнаружить зацикливающиеся пути, которые могут быть немедленно отброшены. Это позволяет обеспечить, чтобы конечные пространства состояний не превращались в бесконечные деревья поиска из-за циклов, но, к сожалению, не дает возможности предотвратить экспоненциальное разрастание нециклических путей в задачах, подобных приведенным на рис. 3.11. Единственный способ предотвращения этого состоит в том, что в памяти нужно хранить больше узлов. В этом заключается фундаментальный компромисс между пространством и временем. Алгоритмы, которые забывают свою историю, обречены на то, чтобы ее повторять.



*Рис. 3.11. Пространства состояний, которые формируют экспоненциально более крупные деревья поиска: пространство состояний, в котором имеются два возможных действия, ведущих от A к B, два — от B к C и т.д.; это пространство состояний содержит  $2^d+1$  состояний, где d — максимальная глубина (а); дерево поиска, которое имеет  $2^d$  ветвей, соответствующих  $2^d$  путям через это пространство (б); пространство состояний в виде прямоугольной решетки (в); состояния, находящиеся в пределах 2 этапов от начального состояния (A), обозначены серым цветом*

Если некоторый алгоритм запоминает каждое состояние, которое он посетил, то может рассматриваться как непосредственно исследующий граф пространства состояний. В частности, можно модифицировать общий алгоритм Tree-Search, чтобы включить в него структуру данных, называемую **закрытым списком**, в котором хранится каждый развернутый узел. (Периферию, состоящую из неразвернутых узлов, иногда называют **открытым списком**.) Если текущий узел совпадает с любым узлом из закрытого списка, то не развертывается, а отбрасывается. Этому новому алгоритму присвоено название алгоритма поиска в графе, Graph-Search (листинг 3.6). При решении задач со многими повторяющимися состояниями алгоритм Graph-Search является намного более эффективным по сравнению с Tree-Search. В наихудшем случае предъявляемые им требования к времени и пространству пропорциональны размеру пространства состояний. Эта величина может оказаться намного меньше, чем  $O(b^d)$ .

Вопрос о том, оптimalен ли поиск в графе, остается сложным. Выше было указано, что появление повторяющегося состояния соответствует обнаружению алгоритмом двух путей в одно и то же состояние. Алгоритм Graph-Search, приведенный в листинге 3.6, всегда отбрасывает вновь обнаруженный путь и оставляет первоначальный; очевидно, что если этот вновь обнаруженный путь короче, чем первоначальный, то алгоритм Graph-Search может упустить оптимальное решение. К счастью,

можно показать (упр. 3.12), что этого не может случиться, если используется либо поиск по критерию стоимости, либо поиск в ширину с постоянными стоимостями этапов; таким образом, эти две оптимальные стратегии поиска в дереве являются также оптимальными стратегиями поиска в графе. При поиске с итеративным углублением, с другой стороны, используется развертывание в глубину, поэтому этот алгоритм вполне может проследовать к некоторому узлу по неоптимальному пути, прежде чем найти оптимальный. Это означает, что при поиске в графе с итеративным углублением необходимо проверять, не является ли вновь обнаруженный путь к узлу лучшим, чем первоначальный, и в случае положительного ответа в нем может потребоваться пересматривать значения глубины и стоимости путей для потомков этого узла.

**Листинг 3.6. Общий алгоритм поиска в графе. Множество *closed* может быть реализовано с помощью хэш-таблицы для обеспечения эффективной проверки повторяющихся состояний. В этом алгоритме предполагается, что первый найденный путь к состоянию *s* является наименее дорогостоящим (см. текст)**

---

```

function Graph-Search(problem, fringe) returns решение
    или индикатор неудачи failure

    closed  $\leftarrow$  пустое множество
    fringe  $\leftarrow$  Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if Empty?(fringe) then return индикатор неудачи failure
        node  $\leftarrow$  Remove-First(fringe)
        if Goal-Test[problem](State[node]) then return
            Solution(node)
        if State[node] не находится в множестве closed then
            добавить State[node] к множеству closed
            fringe  $\leftarrow$  Insert-All(Expand(node, problem), fringe)

```

---

Между прочим, использование закрытого списка *closed* означает, что поиск в глубину и поиск с итеративным углублением больше не имеют линейных требований к пространству. Поскольку в алгоритме Graph-Search каждый узел хранится в памяти, некоторые методы поиска становятся неосуществимыми из-за недостаточного объема памяти.

## 3.6. ПОИСК С ЧАСТИЧНОЙ ИНФОРМАЦИЕЙ

В разделе 3.3 было выдвинуто предположение, что среда является полностью наблюдаемой и детерминированной и что агент имеет информацию о том, каковы последствия каждого действия. Поэтому агент может точно вычислить, какое состояние становится результатом любой последовательности действий, и всегда знает, в каком состоянии он находится. Его восприятия не предоставляют новой информации после выполнения каждого действия. Но что произойдет, если знания о состояниях или действиях являются неполными? Авторы обнаружили, что разные типы неполноты приводят к трем перечисленным ниже типам проблем.

- Проблемы отсутствия датчиков** (называемые также **проблемами совместимости**). Если агент вообще не имеет датчиков, то (насколько ему известно) может находиться в одном из нескольких возможных начальных состояний и поэтому каж-

дое действие способно перевести его в одно из нескольких возможных состояний-преемников.

2. **Проблемы непредвиденных ситуаций.** Если среда наблюдаема лишь частично или действия являются неопределенными, то акты восприятия агента предоставляют новую информацию после выполнения каждого действия. Каждое возможное восприятие определяет непредвиденную ситуацию, к которой необходимо подготовиться с помощью соответствующего плана. Проблема называется **обусловленной сторонним воздействием**, если неопределенность вызвана действиями другого агента.
3. **Проблемы исследования.** Если состояния и действия в среде неизвестны, агент должен действовать так, чтобы их обнаружить. Проблемы исследования могут рассматриваться как крайний случай проблем непредвиденных ситуаций.

В качестве примера мы будем использовать среду мира пылесоса. Напомним, что пространство состояний имеет восемь состояний, как показано на рис. 3.12. Существуют три действия (*Left*, *Right* и *Suck*), и цель состоит в том, чтобы был убран весь мусор (состояния 7 и 8). Если среда наблюдаема, детерминирована и полностью известна, то эта задача решается тривиально с помощью любого из описанных нами алгоритмов. Например, если начальным является состояние 5, то последовательность действий [*Right*, *Suck*] обеспечивает достижение целевого состояния 8. В оставшейся части этого раздела рассматриваются версии данной задачи, в которых отсутствуют датчики и возникают непредвиденные ситуации. Проблемы исследования описаны в разделе 4.5, а проблемы, обусловленные сторонним воздействием, — в главе 6.

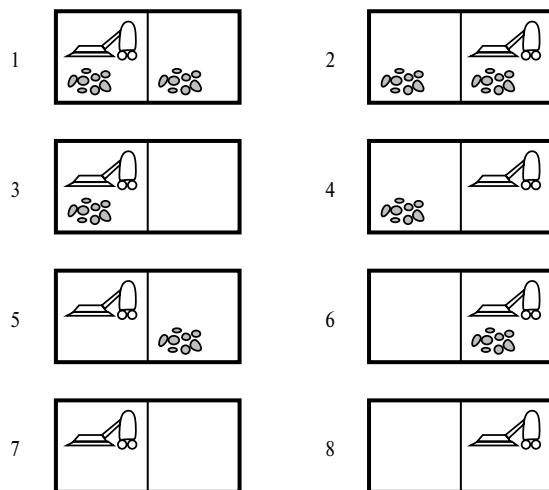


Рис. 3.12. Восемь возможных состояний мира пылесоса

### Проблемы отсутствия датчиков

Предположим, что агенту-пылесосу известны все последствия его действий, но он не имеет датчиков. В таком случае агент знает только, что его начальным состоянием является одно состояние из множества  $\{1, 2, 3, 4, 5, 6, 7, 8\}$ . На первый взгляд можно предположить, что попытки агента предсказать будущую ситуацию окажутся бесполез-

ными, но фактически он может сделать это вполне успешно. Поскольку агент знает, к чему приводят его действия, то может, например, вычислить, что действие *Right* вызовет переход его в одно из состояний {2, 4, 6, 8}, а последовательность действий [*Right*, *Suck*] всегда оканчивается в одном из состояний {4, 8}. Наконец, последовательность действий [*Right*, *Suck*, *Left*, *Suck*] гарантирует достижение целевого состояния 7, независимо от того, каковым является начальное состояние. Мы утверждаем, что агент может ~~и~~ принудительно перевести мир в состояние 7, даже если ему не известно, с какого состояния он начинает. Подведем итог: если мир не является полностью наблюдаемым, то агент должен рассуждать о том, в какое множество состояний (а не в единственное состояние) он может попасть. Мы называем каждое такое множество состояний ~~и~~ доверительным состоянием, поскольку оно показывает, в каких возможных физических состояниях агент может считать себя находящимся в данный момент со всей уверенностью. (В полностью наблюдаемой среде каждое доверительное состояние содержит одно физическое состояние.)

Для решения проблемы отсутствия датчиков необходимо выполнять поиск в пространстве доверительных, а не физических состояний. Первоначальное состояние является доверительным состоянием, а каждое действие становится отображением из одного доверительного состояния в другое. Результат применения некоторого действия к некоторому доверительному состоянию определяется путем объединения результатов применения этого действия к каждому физическому состоянию из этого доверительного состояния. Теперь любой путь объединяет несколько доверительных состояний, а решением является путь, который ведет к такому доверительному состоянию, все члены которого представляют собой целевые состояния. На рис. 3.13 показано пространство достижимых доверительных состояний для детерминированного мира пылесоса без датчиков. Существует только 12 достижимых доверительных состояний, но все пространство доверительных состояний включает каждое возможное множество физических состояний, т.е.  $2^8=256$  доверительных состояний. Вообще говоря, если пространство физических состояний имеет  $S$  состояний, то пространство доверительных состояний имеет  $2^S$  доверительных состояний.

В приведенном выше описании проблем отсутствия датчиков предполагалось, что действия являются детерминированными, но этот анализ, по сути, остается неизменным, если среда — недетерминированная, т.е. если действия могут иметь несколько возможных результатов. Причина этого состоит в том, что в отсутствие датчиков агент не способен определить, какой результат достигнут фактически, поэтому различные возможные результаты становятся просто дополнительными физическими состояниями в доверительном состоянии-преемнике. Например, предположим, что среда подчиняется закону Мэрфи (или закону “подлости”): так называемое действие *Suck* иногда оставляет мусор на полу, но только если на нем еще не было мусора<sup>6</sup>. В таком случае, если действие *Suck* применяется в физическом состоянии 4 (см. рис. 3.12), то существуют два возможных результата: состояния 2 и 4. Теперь применение действия *Suck* в начальном доверительном состоянии, {1, 2, 3, 4, 5, 6, 7, 8}, приводит к доверительному состоянию, представляю-

<sup>6</sup> Мы предполагаем, что большинство читателей сталкиваются с аналогичными проблемами и поэтому выражают сочувствие нашему агенту. Авторы приносят свои извинения владельцам современных, эффективных бытовых приборов, которые не смогут извлечь урок из этого педагогического упражнения.

щему собой объединение множеств результатов для этих восьми физических состояний. Проведя эти вычисления, можно обнаружить, что новым доверительным состоянием снова становится  $\{1, 2, 3, 4, 5, 6, 7, 8\}$ . Таким образом, для агента без датчиков в мире закона Мэрфи действие *Suck* оставляет доверительное состояние неизменным! Это означает, что фактически данная задача неразрешима (см. упр. 3.18). Интуитивно можно понять, что причина этого состоит в том, что агент не может определить, является ли текущий квадрат грязным и поэтому не способен установить, приведет ли действие *Suck* к его очистке или оставит еще больше мусора.

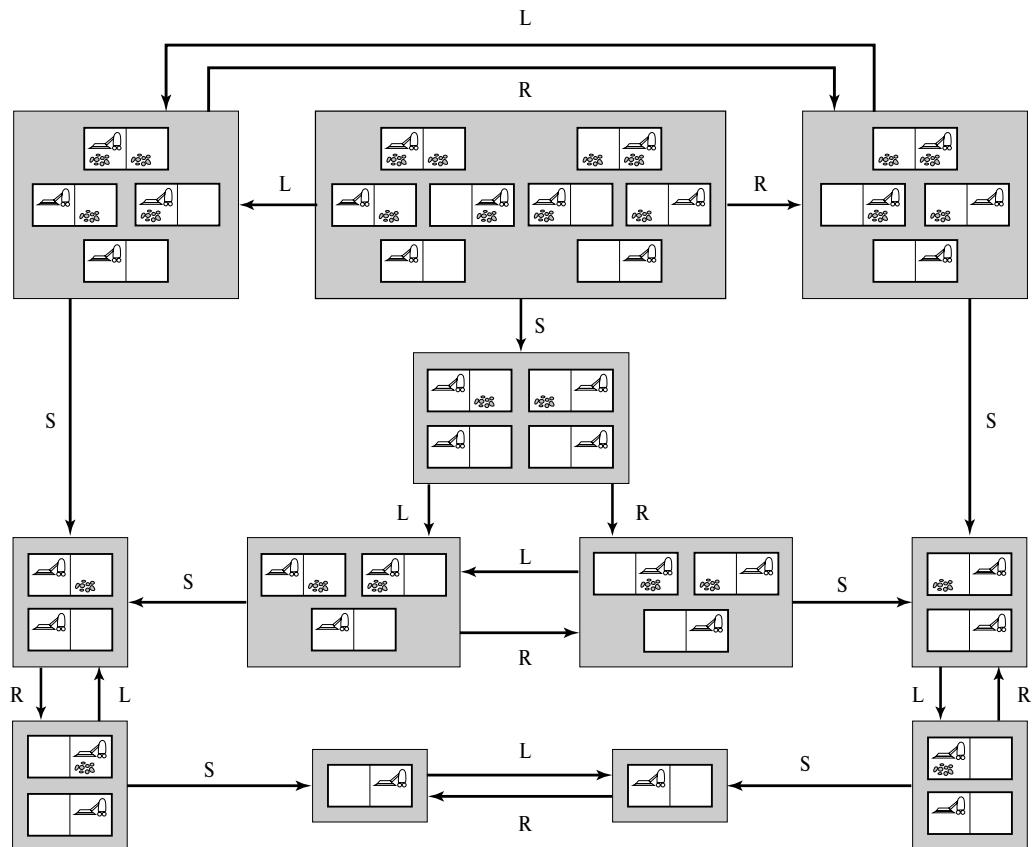


Рис. 3.13. Достижимая часть пространства доверительных состояний для детерминированного мира пылесоса без датчиков. Каждый затененный прямоугольник соответствует одному доверительному состоянию. В любой конкретный момент времени агент находится в одном конкретном доверительном состоянии, но не знает, в каком именно физическом состоянии он находится. Первоначальным доверительным состоянием (с полным незнанием ситуации) является верхний центральный прямоугольник. Действия обозначены дугами с метками, а петли, обозначающие возврат в одно и то же доверительное состояние, опущены для упрощения рисунка

### Проблемы непредвиденных ситуаций

Если среда является таковой, что агент после выполнения действия может получить от своих датчиков в том же состоянии новую информацию, то агент сталкива-

ется с **проблемой непредвиденных ситуаций**. Решение проблемы непредвиденных ситуаций часто принимает форму дерева, в котором каждая ветвь может быть выбрана в зависимости от результатов восприятий, полученных вплоть до этой позиции в дереве. Например, предположим, что агент находится в мире закона Мэрфи и имеет датчик положения и локальный датчик мусора, но не имеет датчика, способного обнаружить мусор в других квадратах. Таким образом, восприятие  $[L, Dirty]$  означает, что агент находится в одном из состояний  $\{1, 3\}$ . Агент может сформулировать последовательность действий  $[Suck, Right, Suck]$ . Всасывание должно было бы перевести текущее состояние в одно из состояний  $\{5, 7\}$ , и тогда передвижение вправо перевело бы данное состояние в одно из состояний  $\{6, 8\}$ . Выполнение заключительного действия *Suck* в состоянии 6 переводит агента в состояние 8, целевое, но выполнение его в состоянии 8 способно снова перевести агента в состояние 6 (согласно закону Мэрфи), и в этом случае данный план оканчивается неудачей.

Исследуя пространство доверительных состояний для этой версии задачи, можно легко определить, что ни одна фиксированная последовательность действий не гарантирует решение данной задачи. Однако решение существует, при условии, что мы не будем настаивать на фиксированной последовательности действий:

```
[ Suck, Right, if [R,Dirty] then Suck ]
```

Тем самым дополняется пространство решений для включения возможности выбора действий с учетом непредвиденных ситуаций, возникающих во время выполнения. Многие задачи в реальном, физическом мире усложняются из-за наличия проблемы непредвиденных ситуаций, поскольку точные предсказания в них невозможны. По этой причине многие люди соблюдают предельную осторожность во время пеших прогулок или вождения автомобиля.

Иногда проблемы непредвиденных ситуаций допускают чисто последовательные решения. Например, рассмотрим полностью наблюдаемый мир закона Мэрфи. Непредвиденные ситуации возникают, если агент выполняет действие *Suck* в чистом квадрате, поскольку при этом в данном квадрате может произойти или не произойти отложение мусора. При том условии, что агент никогда не будет очищать чистые квадраты, не будут возникать какие-либо непредвиденные ситуации и поэтому из любого начального состояния можно будет найти последовательное решение (упр. 3.18).

Алгоритмы для решения проблем непредвиденных ситуаций являются более сложными по сравнению с алгоритмами стандартного поиска, приведенными в этой главе; они рассматриваются в главе 12. Кроме того, проблемы непредвиденных ситуаций требуют применения немного иного проекта агента, в котором агент может действовать до того, как найдет гарантированный план. Это полезно, поскольку вместо предварительного обдумывания каждой возможной непредвиденной ситуации, которая могла бы возникнуть во время выполнения, часто бывает лучше приступить к действию и узнать, какие непредвиденные ситуации действительно возникают. После этого агент может продолжать решать свою задачу, принимая в расчет эту дополнительную информацию. Такой тип **чередования** поиска и выполнения является также полезным при решении проблем исследования (см. раздел 4.5) и при ведении игр (см. главу 6).

## РЕЗЮМЕ

---

В настоящей главе представлены методы, которые могут использоваться агентом для выбора действий в таких вариантах среды, которые являются детерминированными, наблюдаемыми, статическими и полностью известными. В таких случаях агент может формировать последовательности из действий, позволяющих ему достичь своих целей; такой процесс называется **поиском**.

- Прежде чем агент сможет приступить к поиску решений, он должен сформулировать **цель**, а затем использовать эту цель для формулировки **задачи**.
- Задача состоит из четырех частей: **начальное состояние**, множество **действий**, функция **проверки цели** и функция **стоимости пути**. Среда задачи представлена **пространством состояний**, а **путь** через пространство состояний от начального состояния до целевого состояния представляет собой **решение**.
- Для решения любой задачи может использоваться единый, общий алгоритм Tree-Search; конкретные варианты этого алгоритма воплощают различные стратегии.
- Алгоритмы поиска оцениваются на основе **полноты**, **оптимальности**, **временной** и **пространственной сложности**. Сложность зависит от коэффициент ветвления в пространстве состояний,  $b$ , и глубины самого поверхностного решения,  $d$ .
- При **поиске в ширину** для развертывания выбирается самый поверхностный неразвернутый узел в дереве поиска. Этот поиск является полным, оптимальным при единичных стоимостях этапов и характеризуется временной и пространственной сложностью  $O(b^d)$ . В связи с такой пространственной сложностью в большинстве случаев он становится практически не применимым. **Поиск по критерию стоимости** аналогичен поиску в ширину, но предусматривает развертывание узла с самой низкой стоимостью пути,  $g(n)$ . Он является полным и оптимальным, если стоимость каждого шага превышает некоторое положительное предельное значение  $\epsilon$ .
- При **поиске в глубину** для развертывания выбирается самый глубокий неразвернутый узел в дереве поиска. Этот поиск не является ни полным, ни оптимальным, и характеризуется временной сложностью  $O(b^m)$  и пространственной сложностью  $O(bm)$ , где  $m$  — максимальная глубина любого пути в пространстве состояний.
- При **поиске с ограничением глубины** на поиск в глубину налагается установленный предел глубины.
- При **поиске с итеративным углублением** вызывается поиск с ограничением глубины и каждый раз устанавливаются увеличивающиеся пределы, до тех пор, пока цель не будет найдена. Этот поиск является полным и оптимальным при единичных стоимостях этапов и характеризуется временной сложностью  $O(b^d)$  и пространственной сложностью  $O(bd)$ .
- **Двунаправленный поиск** способен чрезвычайно уменьшить временную сложность, но он не всегда применим и может потребовать слишком много пространства.

- Если пространство состояний представляет собой граф, а не дерево, то может оказаться целесообразной проверка повторяющихся состояний в дереве поиска. Алгоритм Graph-Search устраняет все дублирующие состояния.
- Если среда является частично наблюдаемой, то агент может применить алгоритмы поиска в пространстве **доверительных состояний**, или множестве возможных состояний, в которых может находиться агент. В некоторых случаях может быть сформирована единственная последовательность решения; в других случаях агенту требуется **план действий в непредвиденных ситуациях**, чтобы иметь возможность справиться со всеми неизвестными обстоятельствами, которые могут возникнуть.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Большинство задач поиска в пространстве состояний, проанализированных в этой главе, имеют длинную историю, отраженную в литературе, и являются менее тривиальными, чем может показаться на первый взгляд. Задача с миссионерами и каннибалами, используемая в упр. 3.9, была подробно проанализирована Амарелем [24]. До него эту задачу ввели в проблематику искусственного интеллекта Саймон и Ньюэлл [1420], и в проблематику исследования операций — Беллман и Дрейфус [96]. Работы наподобие проведенных Ньюэллом и Саймоном над программами Logic Theorist [1127] и GPS [1129] привели к тому, что алгоритмы поиска считались основным оружием в арсенале исследователей искусственного интеллекта 1960-х годов, а сами процессы решения задач стали рассматриваться в качестве канонической проблемы искусственного интеллекта. К сожалению, в то время в области автоматизации этапа формулировки задачи было предпринято слишком мало усилий. Более современная трактовка подхода к представлению и абстрагированию задач, включая применение программ искусственного интеллекта, которые (отчасти) сами выполняют эти функции, изложена в книге Кноблока [807].

Задача игры в восемь — это “младшая сестра” задачи игры в пятнадцать, которая была изобретена знаменитым американским разработчиком игр Сэмом Ллойдом [955] в 1870-х годах. Игра в пятнадцать быстро приобрела в Соединенных Штатах огромную популярность, которую можно сравнить лишь с более современной сенсацией, вызванной изобретением кубика Рубика. Она также быстро привлекла внимание математиков [739], [1486]. Редакторы *American Journal of Mathematics* заявили: “Игра в пятнадцать в последние несколько недель занимает все внимание американской публики, и можно с уверенностью сказать, что ею интересуются девять из десяти людей всех полов и возрастов и всех общественных положений. Но не это побудило нас, редакторов, включить статьи, посвященные этой теме, в наш журнал *American Journal of Mathematics*, а тот факт, что...” (далее следует краткое описание аспектов игры в пятнадцать, представляющих интерес с точки зрения математики). Исчерпывающий анализ игры в восемь был проведен с помощью компьютера Шоффилдом [1363]. Ратнер и Уормут [1268] показали, что общая версия игры (не в пятнадцать, а в  $n \times n - 1$ ) принадлежит к классу NP-полных задач.

Задача с восемью ферзями была впервые опубликована анонимно в немецком шахматном журнале *Schach* в 1848 году; позднее ее создание приписали некоему

Максу Бесселю. Она была повторно опубликована в 1850 году и на этот раз привлекала внимание выдающегося математика Карла Фридриха Гаусса, который попытался перечислить все возможные решения, но нашел только 72. Наук (Nauck) опубликовал все 92 решения позднее, в том же 1850 году. Нетто [1121] обобщил эту задачу до  $n$  ферзей, а Абрамсон и Янг [2] нашли алгоритм с оценкой  $O(n)$ .

Каждая из реальных задач поиска, перечисленных в данной главе, была предметом значительных усилий исследователей. Методы выбора оптимальных расписаний авиаперелетов по большей части остаются недоступными для общего пользования (закрытыми), но Карл де Маркен (Carl de Marcken) сообщил авторам (в личной беседе), что структуры формирования цен на авиабилеты и связанные с ними ограничения стали настолько сложными, что задача выбора оптимального авиарейса является формально неразрешимой. Задача коммивояжера (Traveling Salesperson Problem — TSP) — это стандартная комбинаторная проблема в теоретических компьютерных науках [898], [899]. Кэрп [772] доказал, что задача TSP является NP-трудной, но для нее были разработаны эффективные методы эвристической аппроксимации [933]. Арора [41] разработал полностью полиномиальную схему аппроксимации для евклидовых вариантов задачи TSP. Обзор методов компоновки СБИС был сделан Шахукаром и Мазумдером [1390], а в журналах по сверхбольшим интегральным микросхемам (СБИС) появилось много статей, посвященных оптимизации компоновки. Задачи робототехнической навигации и сборки обсуждаются в главе 25.

Алгоритмы неинформированного поиска для решения задач являются центральной темой классических компьютерных наук [680] и исследования операций [417]; более современные результаты исследований приведены в книгах Део и Панга [390], а также Галло и Паллоттино [516]. Метод поиска в ширину применительно к решению задач с лабиринтами был сформулирован Муром [1078]. Метод **динамического программирования** [96], в котором предусматривается систематическая регистрация решений для всех подзадач с возрастающей длиной, может рассматриваться как форма поиска в ширину в графах. Алгоритм определения кратчайшего пути между двумя точками, предложенный Дейкстрой [399], является предшественником алгоритма поиска по критерию стоимости.

Одна из версий алгоритма поиска с итеративным углублением, предназначенная для эффективного ведения игры в шахматы с контролем времени, была впервые применена Слейтом и Аткином [1429] в программе ведения шахматной игры Chess 4.5, но ее применению для поиска кратчайшего пути в графе мы обязаны Корфу [835]. В некоторых случаях может также оказаться очень эффективным двунаправленный поиск, который был предложен Полом [1219], [1221].

Частично наблюдаемые и недетерминированные варианты среды не были достаточно глубоко исследованы в этом подходе к решению задач. Некоторые проблемы эффективности при поиске в пространстве доверительных состояний рассматривались Генезеретом и Нурбакшем [538]. Кёниг и Симмонс [819] изучали навигацию робота из неизвестной начальной позиции, а Эрдман и Мэйсон [439] исследовали проблему робототехнического манипулирования без датчиков, используя непрерывную форму поиска в пространстве доверительных состояний. Поиск в условиях неопределенных ситуаций изучался в этой подобласти планирования (см. главу 12). По большей части в подходе к изучению планирования и осуществления действий с неопределенной информацией используются инструментальные средства теории вероятностей и теории решений (см. главу 17).

Книги Нильссона [1141], [1142] являются хорошим общим источником информации о классических алгоритмах поиска. Исчерпывающий и более современный обзор можно найти в [838]. Статьи о новых алгоритмах поиска (которые, как это ни удивительно, продолжают изобретаться) публикуются в таких журналах, как *Artificial Intelligence*.

## УПРАЖНЕНИЯ

- 3.1. Самостоятельно сформулируйте определения следующих понятий: состояние, пространство состояний, дерево поиска, поисковый узел, цель, действие, функция определения преемника и коэффициент ветвления.
- 3.2. Объясните, почему составление формулировки задачи должно осуществляться вслед за составлением формулировки цели.
- 3.3. Предположим, что выражение  $\text{Legal-Actions}(s)$  обозначает множество действий, которые являются допустимыми в состоянии  $s$ , а выражение  $\text{Result}(a, s)$  обозначает состояние, которое следует из выполнения допустимого действия  $a$  в состоянии  $s$ . Определите функцию  $\text{Successor-Fn}$  в терминах выражений  $\text{Legal-Actions}$  и  $\text{Result}$ , и наоборот.
- 3.4. Покажите, что в задаче игры в восемь состояния подразделяются на два непересекающихся множества, таких, что ни одно состояние из первого множества не может быть преобразовано в состояние из второго множества, даже с применением сколь угодно большого количества ходов. (*Подсказка.* См. [102].) Разработайте процедуру, позволяющую узнать, к какому множеству относится данное состояние, и объясните, для чего нужно иметь под рукой такую процедуру, формируя состояния случайным образом.
- 3.5. Рассмотрите задачу с  $n$  ферзями, используя “эффективную” инкрементную формулировку, приведенную на с. 119. Объясните, почему размер пространства состояний равен по меньшей мере  $\sqrt[3]{n!}$ , и оцените наибольшее значение  $n$ , для которого является осуществимым исчерпывающее исследование. (*Подсказка.* Определите нижнюю границу коэффициента ветвления, рассматривая максимальное количество клеток, которые ферзь может атаковать в любом столбце.)
- 3.6. Всегда ли наличие конечного пространства состояний приводит к получению конечного дерева поиска? А что можно сказать о конечном пространстве состояний, которое является деревом? Можете ли вы указать более точно, применение пространств состояний каких типов всегда приводит к получению конечных деревьев поиска? (*Адаптировано из [99].*)
- 3.7. Укажите для каждой из следующих задач ее компоненты: начальное состояние, проверку цели, функцию определения преемника и функцию стоимости. Выберите формулировку, которая является достаточно точной, чтобы ее можно было успешно реализовать.
  - а) Необходимо раскрасить плоскую карту, используя только четыре цвета, таким образом, чтобы никакие два смежных региона не имели один и тот же цвет.

- 6) Обезьяна, имеющая рост 3 фута, находится в комнате, где под потолком, на высоте 8 футов, подвешено несколько бананов. Обезьяна хочет получить бананы. В комнате находятся две проволочные корзины высотой по 3 фута каждая, которые можно передвигать, ставить друг на друга и на которые можно залезать.
- б) Некоторая программа выводит сообщение “недопустимая входная запись” после передачи ей некоторого файла, состоящего из входных записей. Известно, что обработка каждой записи происходит независимо от других записей. Требуется обнаружить, какая запись является недопустимой.
- г) Имеются три кувшина, с емкостью 12 галлонов, 8 галлонов и 3 галлона, а также водопроводный кран. Кувшины можно заполнять или опорожнять, выливая воду из одного кувшина в другой или на землю. Необходимо отмерить ровно один галлон.
- 3.8. Рассмотрите пространство состояний, в котором начальным состоянием является число 1, а функция определения преемника для состояния  $n$  возвращает два состояния: числа  $2n$  и  $2n+1$ .
- Нарисуйте часть пространства состояний, которая относится к состояниям 1–15.
  - Предположим, что целевым состоянием является 11. Перечислите последовательности, в которых будут посещаться узлы при поиске в ширину, поиске с ограничением глубины, с пределом 3, и поиске с итеративным углублением.
  - Будет ли подходящим для решения этой задачи двунаправленный поиск? Если да, то опишите подробно, как действовал бы этот метод поиска.
  - Каковым является коэффициент ветвления в каждом направлении двунаправленного поиска?
  - Не подсказывает ли вам ответ на вопрос упр. 3.8, в, что нужно найти другую формулировку этой задачи, которая позволила бы решить проблему перехода из состояния 1 в заданное целевое состояние почти без поиска?
- 3.9.  Задача с **миссионерами и каннибалами** обычно формулируется следующим образом. Три миссионера и три каннибала находятся на одной стороне реки, где также находится лодка, которая может выдержать одного или двух человек. Найдите способ перевезти всех на другой берег реки, никогда не оставляя где-либо группу миссионеров, которую превосходила бы по численности группа каннибалов. Это — известная задача в искусственном интеллекте, поскольку она была темой первой статьи, в которой был применен подход к формулировке проблемы с аналитической точки зрения [24].
- Точно сформулируйте эту задачу, определяя только те различия, которые необходимы для обеспечения правильного решения. Нарисуйте схему полного пространства состояний.
  - Реализуйте и решите эту задачу оптимальным образом, используя соответствующий алгоритм поиска. Действительно ли имеет смысл проверять наличие повторяющихся состояний?
  - Почему, по вашему мнению, люди сталкиваются с затруднениями при решении этой головоломки, несмотря на то, что пространство ее состояний является чрезвычайно простым?

- 3.10.** Реализуйте следующие две версии функции определения преемника для задачи игры в восемь. Первая из них должна формировать сразу всех преемников, копируя и редактируя структуру данных игры в восемь, а вторая при каждом ее вызове должна формировать по одному новому преемнику и действовать по принципу непосредственной модификации родительского состояния (отменяя эти модификации в случае необходимости). Напишите версии процедуры поиска в глубину с итеративным углублением, в которых используются эти функции, и сравните данные об их производительности.
- 3.11.** На с. 135 упоминался **поиск с итеративным удлинением**, итеративный аналог поиска по критерию стоимости. Его идея состоит в том, что должны использоваться увеличивающиеся пределы стоимости пути. Если сформирован узел, стоимость пути для которого превышает текущий предел, этот узел немедленно отбрасывается. При каждой новой итерации предел устанавливается равным самому низкому значению стоимости пути для любого узла, отвергнутого в предыдущей итерации.
- Покажите, что этот алгоритм является оптимальным применительно к общим методам определения стоимости пути.
  - Рассмотрите однородное дерево с коэффициентом ветвления  $b$ , глубиной решения  $d$  и единичными стоимостями этапов. Какое количество итераций требуется при итеративном удлинении?
  - Рассмотрите стоимости этапов, взятые из непрерывного диапазона  $[0, 1]$  с минимальной положительной стоимостью  $\varepsilon$ . Сколько итераций потребуется в самом неблагоприятном случае?
  - Реализуйте этот алгоритм и примените его к экземплярам задачи игры в восемь и задачи коммивояжера. Сравните производительность этого алгоритма с производительностью алгоритма поиска по критерию стоимости и прокомментируйте полученные результаты.
- 3.12.** Докажите, что поиск по критерию стоимости и поиск в ширину с постоянными значениями стоимости этапа являются оптимальными при их использовании с алгоритмом Graph-Search. Продемонстрируйте такое пространство состояний с постоянными значениями стоимости этапа, в котором алгоритм Graph-Search с использованием итеративного углубления находит неоптимальное решение.
- 3.13.** Опишите пространство состояний, в котором поиск с итеративным углублением характеризуется гораздо более низкой производительностью по сравнению с поиском в глубину (например,  $O(n^2)$ , в отличие от  $O(n)$ ).
- 3.14.** Напишите программу, которая принимает в качестве входных данных URL-локаторы двух Web-страниц и находит путь от одной к другой, состоящий из ссылок. В чем состоит подходящая для этого стратегия поиска? Действительно ли имеет смысл применять двунаправленный поиск? Можно ли использовать для реализации функции определения преемника машину поиска?
- 3.15.** Рассмотрите задачу определения кратчайшего пути между двумя точками на плоскости, на которой имеются препятствия в виде выпуклых многоугольников, как показано на рис. 3.14. Это — идеализация задачи, которую должен

решать робот, прокладывая свой путь через среду, в которой очень мало свободного места.

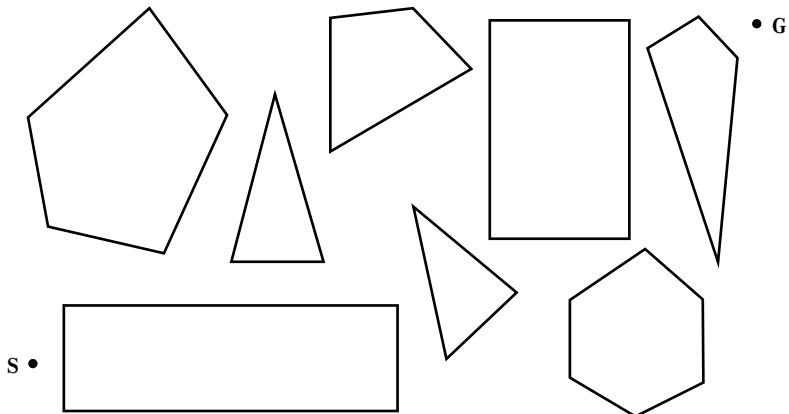


Рис. 3.14. Сцена с многоугольными препятствиями

- Предположим, что пространство состояний состоит из всех позиций  $(x, y)$  на плоскости. Каково количество состояний в этом пространстве? Каково в нем количество путей к цели?
  - Объясните в нескольких словах, почему в этой двухмерной сцене кратчайший путь от одной вершины многоугольника до любой другой должен состоять из прямолинейных отрезков, соединяющих некоторые вершины многоугольников. Теперь определите более приемлемое пространство состояний. Насколько велико это пространство состояний?
  - Определите функции, необходимые для реализации решения этой задачи поиска, включая функцию определения преемника, которая принимает в качестве входных данных координаты любой из вершин и возвращает множество вершин, достижимых по прямой линии от данной вершины. (Не забывайте при этом о соседних вершинах того же многоугольника.) Используйте в качестве значения эвристической функции расстояние между указанными точками по прямой.
  - Примените для решения ряда задач из этой области один или несколько алгоритмов, представленных в настоящей главе, и прокомментируйте данные об их производительности.
- 3.16.** Определение задачи обеспечения навигации робота, приведенной в упр. 3.15, можно преобразовать в определение среды следующим образом.

- Акт восприятия будет представлять собой список позиций видимых вершин, сформированный относительно агента. Акт восприятия не предусматривает определение позиции робота! Робот должен определять свою собственную позицию по карте; на данный момент можно предположить, что каждое местоположение имеет другое “представление”.
- Каждое действие должно быть вектором, описывающим прямолинейный путь, по которому будет следовать робот. Если путь свободен, то действие выполняется успешно; в противном случае робот останавливается в той точке,

где его путь впервые сталкивается с препятствием. Если агент возвращает нулевой вектор движений и находится в целевой позиции (которая является постоянной и известной), то среда должна телепортировать этого агента в случайно выбранное местоположение (не находящееся в пределах препятствия).

- Показатель производительности предусматривает штрафование агента на 1 пункт за каждую единицу пройденного расстояния и награждение его 1000 пунктами каждый раз после достижения цели.

Ниже перечислены предлагаемые задания.

- а) Реализуйте описанную среду и агента, решающего задачи для этой среды. После каждой телепортации агент должен будет сформулировать новую задачу, которая предусматривает также определение его текущего местоположения.
  - б) Зарегистрируйте данные о производительности предложенного вами агента (для этого предусмотрите выработку агентом соответствующих комментариев по мере его передвижения в среде) и составьте отчет о его производительности по данным, охватывающим больше 100 эпизодов.
  - в) Модифицируйте среду так, чтобы в 30% случаев движение агента заканчивались в не предусмотренном им месте назначения (выбранном случайным образом среди других видимых вершин, если таковые имеются, а в противном случае соответствующем ситуации, в которой вообще не было выполнено никакого движения). Это — грубая модель ошибок при выполнении движений реального робота. Доработайте определение агента так, чтобы при обнаружении указанной ошибки он определял, где находится, а затем создавал план возвращения в то место, где он находился прежде, и возобновлял выполнение прежнего плана. Помните, что иногда попытка возвращения в прежнее место также может оканчиваться неудачей! Продемонстрируйте пример агента, который успешно преодолевает две последовательные ошибки движения и все равно достигает цели.
  - г) Опробуйте две различные схемы возобновления работы после ошибки: во-первых, отправиться к ближайшей вершине из первоначального маршрута и, во-вторых, перепланировать маршрут к цели от нового местоположения. Сравните производительность всех схем возобновления работы. Влияет ли на результаты такого сравнения включение затрат на поиск?
  - д) Теперь предположим, что есть такие местоположения, представления среды из которых являются идентичными. (Например, предположим, что мир — это решетка с квадратными препятствиями.) С какой проблемой теперь сталкивается агент? Как должны выглядеть решения?
- 3.17. На с. 115 было указано, что мы не будем принимать во внимание задачи с отрицательными значениями стоимости пути. В данном упражнении эта тема рассматривается немного более подробно.
- а) Предположим, что действия могут иметь произвольно большие отрицательные стоимости; объясните, почему такая ситуация может вынудить любой оптимальный алгоритм исследовать полное пространство состояний.
  - б) Удастся ли выйти из этого положения, потребовав, чтобы стоимости этапов были больше или равны некоторой отрицательной константе  $c$ ? Рассмотрите и деревья, и графы.

- в)** Предположим, что имеется множество операторов, образующих цикл, так что выполнение операторов этого множества в определенном порядке не приводит к какому-либо чистому изменению состояния. Если все эти операторы имеют отрицательную стоимость, то какие выводы из этого следуют применительно к оптимальному поведению агента в такой среде?
- г)** Можно легко представить себе, что операторы с высокой отрицательной стоимостью имеются даже в таких проблемных областях, как поиск маршрута. Например, некоторые участки дороги могут оказаться настолько живописными, что стремление ознакомиться с ними намного перевесит обычные здравые рассуждения о стоимости, измеряемой в терминах затрат времени и топлива. Объясните, применяя точные термины, принятые в контексте поиска в пространстве состояний, почему все же люди не ведут свои автомобили неопределенно долго по живописным циклическим участкам пути, и укажите, каким образом нужно определить пространство состояний и операторы для задачи поиска маршрута, чтобы агенты с искусственным интеллектом также могли избежать попадания в цикл.
- д)** Можете ли вы придумать пример такой реальной проблемной области, в которой стоимости этапов таковы, что могут вызвать возникновение цикла?
- 3.18.** Рассмотрим мир пылесоса без датчиков, с двумя местоположениями, подчиняющийся закону Мэрфи. Нарисуйте пространство доверительных состояний, достижимых из начального доверительного состояния  $\{1, 2, 3, 4, 5, 6, 7, 8\}$ , и объясните, почему эта задача неразрешима. Покажите также, что если бы этот мир был полностью наблюдаемым, то существовала бы последовательность решения для каждого возможного начального состояния.
- 3.19.** Рассмотрим задачу в мире пылесоса, который определен на рис. 2.2.
- Какой из алгоритмов, определенных в этой главе, был бы подходящим для решения этой задачи? Должен ли этот алгоритм проверять наличие повторяющихся состояний?
  - Примените выбранный вами алгоритм для вычисления оптимальной последовательности действий в мире с размером  $3 \times 3$ , в начальном состоянии которого в трех верхних квадратах имеется мусор, а агент находится в центре.
  - Сконструируйте агента, выполняющего поиск в этом мире пылесоса, и оцените его работу в множестве миров с размером  $3 \times 3$ , характеризующемся вероятностью наличия мусора в каждом квадрате, равной 0.2. Включите в состав показателя производительности не только стоимость поиска, но и стоимость пути, используя для них подходящий “курс обмена”.
  - Сравните вашего лучшего поискового агента с простым рандомизированным рефлексным агентом, который всасывает мусор, если последний имеется, а в противном случае выполняет случайно выбранные перемещения.
  - Рассмотрите, что произошло бы, если бы мир расширился до размеров  $n \times n$ . Как производительность данного поискового агента и рефлексного агента зависит от  $n$ ?

# 4 ИНФОРМИРОВАННЫЙ ПОИСК И ИССЛЕДОВАНИЕ ПРОСТРАНСТВА СОСТОЯНИЙ

*В этой главе показано, как можно с помощью информации о пространстве состояний не дать алгоритмам заблудиться в темноте.*

В главе 3 показано, что неинформированные стратегии поиска позволяют находить решения задач путем систематической выработки новых состояний и их проверки применительно к цели. К сожалению, в большинстве случаев эти стратегии являются чрезвычайно неэффективными. Как показано в настоящей главе, информированные стратегии поиска (в которых используются знания, относящиеся к конкретной задаче) обеспечивают более эффективный поиск решения. В разделе 4.1 описаны информированные версии алгоритмов главы 3, а в разделе 4.2 показано, как может быть получена необходимая информация, относящаяся к конкретной задаче. В разделах 4.3 и 4.4 представлены алгоритмы, которые выполняют исключительно **локальный поиск** в пространстве состояний, оценивая и модифицируя одно или несколько текущих состояний вместо систематического исследования путей из начального состояния. Эти алгоритмы применимы для решения задач, в которых стоимость пути не представляет интереса и требуется лишь найти состояние, соответствующее решению. К этому семейству алгоритмов локального поиска относятся методы, созданные под влиянием исследований в области статистической физики (**моделируемый отжиг**) и эволюционной биологии (**генетические алгоритмы**). Наконец, в разделе 4.5 рассматривается **поиск в оперативном режиме**, в котором агент сталкивается с полностью неизвестным пространством состояний.

## 4.1. СТРАТЕГИИ ИНФОРМИРОВАННОГО (ЭВРИСТИЧЕСКОГО) ПОИСКА

В данном разделе показано, как стратегия **информированного поиска** (в которой кроме определения самой задачи используются знания, относящиеся к данной конкретной проблемной области) позволяет находить решения более эффективно, чем стратегия неинформированного поиска.

Общий рассматриваемый здесь подход называется **поиском по первому наилучшему совпадению**. Поиск по первому наилучшему совпадению представляет собой разновидность общего алгоритма Tree-Search или Graph-Search, в котором узел для развертывания выбирается на основе **функции оценки**,  $f(n)$ . По традиции для развертывания выбирается узел с наименьшей оценкой, поскольку такая оценка измеряет расстояние до цели. Поиск по первому наилучшему совпадению может быть реализован в рамках описанной в данной книге общей инфраструктуры поиска с помощью очереди по приоритету — структуры данных, в которой периферия поиска поддерживается в возрастающем порядке  $f$ -значений.

Название “поиск по первому наилучшему совпадению” (best first search) узаконено традицией, но неточно. В конце концов, если бы мы действительно могли развертывать наилучший узел первым, то не было бы и поиска как такового; решение задачи представляло бы собой прямое шествие к цели. Единственное, что мы можем сделать, — это выбрать узел, который представляется наилучшим в соответствии с функцией оценки. Если функция оценки действительно является точной, то выбранный узел в самом деле окажется наилучшим узлом, но фактически функция оценки иногда оказывается малопригодной и способной завести поиск в тупик. Тем не менее авторы будут придерживаться названия “поиск по первому наилучшему совпадению”, поскольку более подходящее название “поиск по первому совпадению, которое можно считать наилучшим” было бы довольно громоздким.

Существует целое семейство алгоритмов поиска по первому наилучшему совпадению, Best-First-Search, с различными функциями оценки<sup>1</sup>. Ключевым компонентом этих алгоритмов является **эвристическая функция**<sup>2</sup>, обозначаемая как  $h(n)$ :

$$h(n) = \text{оценка стоимости наименее дорогостоящего пути от узла } n \text{ до целевого узла}$$

Например, в задаче поиска маршрута в Румынии можно было бы оценивать стоимость наименее дорогостоящего пути от Арада до Бухареста с помощью расстояний по прямой до Бухареста, измеряемых в узловых точках маршрута от Арада до Бухареста.

Эвристические функции (или просто эвристики) представляют собой наиболее общую форму, в которой к алгоритму поиска подключаются дополнительные знания о задаче. Эвристики рассматриваются более подробно в разделе 4.2, а на данный момент мы будем определять их как произвольные функции, относящиеся к конкретной проблеме, с одним ограничением: если  $n$  — целевой узел, то  $h(n)=0$ . В остав-

<sup>1</sup> В упр. 4.3 предложено показать, что это семейство включает несколько знакомых читателю неинформированных алгоритмов.

<sup>2</sup> Эвристическая функция  $h(n)$  принимает в качестве входного параметра некоторый узел, но зависит только от состояния данного узла.

шейся части настоящего раздела рассматриваются два способа использования эвристической информации для управления поиском.

### Жадный поиск по первому наилучшему совпадению

При **жадном поиске по первому наилучшему совпадению**<sup>3</sup> предпринимаются попытки развертывания узла, который рассматривается как ближайший к цели на том основании, что он со всей вероятностью должен быстро привести к решению. Таким образом, при этом поиске оценка узлов производится с использованием только эвристической функции:  $f(n) = h(n)$ .

Теперь рассмотрим, как используется этот алгоритм при решении задачи поиска маршрута в Румынии на основе эвристической функции определения **расстояния по прямой** (Straight Line Distance — SLD), для которой принято обозначение  $h_{SLD}$ . Если целью является Бухарест, то необходимо знать расстояния по прямой от каждого прочего города до Бухареста, которые приведены в табл. 4.1. Например,  $h_{SLD}(In(Arad)) = 366$ . Обратите внимание на то, что значения  $h_{SLD}$  не могут быть вычислены на основании описания самой задачи. Кроме того, для использования этой эвристической функции нужен определенный опыт, позволяющий узнать, каким образом значения  $h_{SLD}$  связаны с действительными дорожными расстояниями, а это означает, что данная функция исходит из практики.

Таблица 4.1. Значения  $h_{SLD}$  — расстояния по прямой до Бухареста

Обозначение узла	Название города	Расстояние по прямой до Бухареста	Обозначение узла	Название города	Расстояние по прямой до Бухареста
Arad	Арад	366	Mehadia	Мехадия	241
Bucharest	Бухарест	0	Neamt	Нямц	234
Craiova	Крайова	160	Oradea	Орадя	380
Drobeta	Дробета	242	Pitesti	Питеши	100
Eforie	Эфорие	161	Rimnicu Vilcea	Рымнику-Вылча	193
Fagaras	Фэгэраш	176	Sibiu	Сибиу	253
Giurgiu	Джурджу	77	Timisoara	Тимишоара	329
Hirsova	Хыршова	151	Urziceni	Урзичени	80
Iasi	Яссы	226	Vaslui	Васлуй	199
Lugoj	Лугож	244	Zerind	Зеринд	374

На рис. 4.1 показан процесс применения жадного поиска по первому наилучшему совпадению с использованием значений  $h_{SLD}$  для определения пути от Арада до Бухареста. Первым узлом, подлежащим развертыванию из узла *Arad*, является узел *Sibiu*, поскольку город Сибиу находится ближе к Бухаресту, чем города Зеринд или Тимишоара. Следующим узлом, подлежащим развертыванию, является узел *Fagaras*, поскольку теперь ближайшим к Бухаресту является город Фэгэраш. Узел

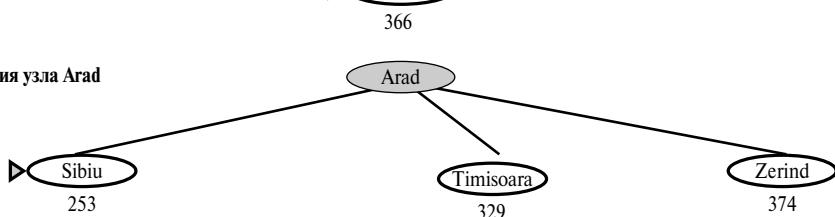
<sup>3</sup> В первом издании данной книги этот алгоритм поиска именовался просто **жадным поиском**; в книгах других авторов для него применяется название **поиск по первому наилучшему совпадению**. Авторы настоящей книги используют последний термин в более общем смысле, в соответствии с трактовкой Перла [1188].

*Fagaras*, в свою очередь, обеспечивает формирование узла *Bucharest*, который является целевым. Применение в процессе решения данной конкретной задачи алгоритма жадного поиска по первому наилучшему совпадению с использованием функции  $h_{SLD}$  позволяет найти решение без развертывания какого-либо узла, не находящегося в пути решения; это означает, что стоимость такого поиска является минимальной. Но само найденное решение не оптимально: путь до Бухареста через города Сибиу и Фэгэраш на 32 километра длиннее, чем путь через города Рымнику-Вылча и Питешти. Это замечание показывает, почему данный алгоритм называется “жадным”: на каждом этапе он пытается подойти к цели как можно ближе (фигурально выражаясь, “захватить как можно больше”).

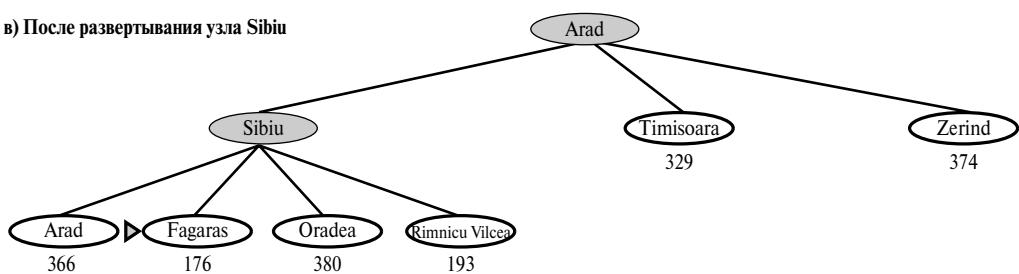
а) Начальное состояние



б) После развертывания узла Arad



в) После развертывания узла Sibiu



г) После развертывания узла Fagaras

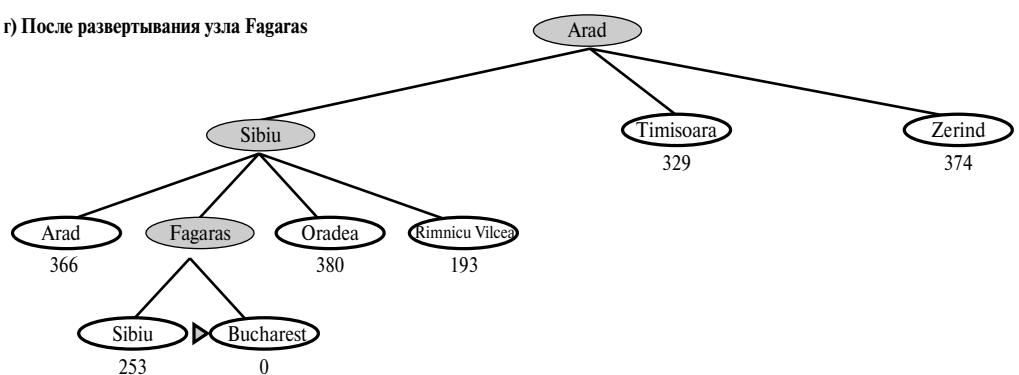


Рис. 4.1. Этапы жадного поиска пути до Бухареста по первому наилучшему совпадению с использованием эвристической функции  $h_{SLD}$ , определяющей расстояние по прямой. Узлы обозначены с помощью их  $h$ -значений

Процедура минимизации  $h(n)$  восприимчива к фальстартам (при ее использовании иногда приходится отменять начальные этапы). Рассмотрим задачу поиска пути

от города Яссы до города Фэгэраш. Эта эвристическая функция подсказывает, что в первую очередь должен быть развернут узел города Нямц, *Neamt*, поскольку он является ближайшим к узлу *Fagaras*, но этот путь становится тупиковым. Решение состоит в том, чтобы отправиться вначале в город Васлуй (а этот этап, согласно данной эвристической функции, фактически уводит дальше от цели), а затем продолжать движение через Урзичени, Бухарест и, наконец, в Фэгэраш. Поэтому в данном случае применение указанной эвристической функции вызывает развертывание ненужных узлов. Более того, если не будет предусмотрено обнаружение повторяющихся состояний, то решение так никогда не будет найдено — процедура поиска станет совершать возвратно-поступательные движения между узлами *Neamt* и *Iasi*.

Жадный поиск по первому наилучшему совпадению напоминает поиск в глубину в том отношении, что этот алгоритм предпочитает на пути к цели постоянно следовать по единственному пути, но возвращается к предыдущим узлам после попадания в тупик. Данный алгоритм страдает от тех же недостатков, что и алгоритм поиска в глубину: он не является оптимальным, к тому же он — не полный (поскольку способен отправиться по бесконечному пути, да так и не вернуться, чтобы опробовать другие возможности). При этом в наихудшем случае оценки временной и пространственной сложности составляют  $O(b^m)$ , где  $m$  — максимальная глубина пространства поиска. Однако хорошая эвристическая функция позволяет существенно сократить такую сложность. Величина этого сокращения зависит от конкретной задачи и от качества эвристической функции.

### Поиск A\*: минимизация суммарной оценки стоимости решения

Наиболее широко известная разновидность поиска по первому наилучшему совпадению называется **поиском A\*** (читается как “А звездочка”). В нем применяется оценка узлов, объединяющая в себе  $g(n)$ , стоимость достижения данного узла, и  $h(n)$ , стоимость прохождения от данного узла до цели:

$$f(n) = g(n) + h(n)$$

Поскольку функция  $g(n)$  позволяет определить стоимость пути от начального узла до узла  $n$ , а функция  $h(n)$  определяет оценку стоимости наименее дорогостоящего пути от узла  $n$  до цели, то справедлива следующая формула:

$$f(n) = \text{оценка стоимости наименее дорогостоящего пути решения, проходящего через узел } n$$

Таким образом, при осуществлении попытки найти наименее дорогостоящее решение, по-видимому, разумнее всего вначале попытаться проверить узел с наименьшим значением  $g(n) + h(n)$ . Как оказалось, данная стратегия является больше чем просто разумной: если эвристическая функция  $h(n)$  удовлетворяет некоторым условиям, то поиск A\* становится и полным, и оптимальным.

Анализ оптимальности поиска A\* является несложным, если этот метод используется в сочетании с алгоритмом Tree-Search. В таком случае поиск A\* является оптимальным, при условии, что  $h(n)$  представляет собой **допустимую эвристическую функцию**, т.е. при условии, что  $h(n)$  никогда не переоценивает стоимость достижения цели. Допустимые эвристические функции являются по своей сути оптимистическими функциями, поскольку возвращают значения стоимости решения за-

дачи, меньшие по сравнению с фактическими значениями стоимости. А поскольку  $g(n)$  — точная стоимость достижения узла  $n$ , из этого непосредственно следует, что функция  $f(n)$  никогда не переоценивает истинную стоимость достижения решения через узел  $n$ .

Очевидным примером допустимой эвристической функции является функция определения расстояния по прямой  $h_{SLD}$ , которая уже использовалась в данной главе для поиска пути в Бухарест. Расстояние по прямой является допустимым, поскольку кратчайший путь между любыми двумя точками лежит на прямой; это означает, что длина прямого пути по определению не может представлять собой переоценку длины пути. На рис. 4.2 показан процесс поиска A\* пути в Бухарест с помощью дерева. Значения  $g$  вычисляются на основании стоимостей этапов, показанных на рис. 3.1, а значения  $h_{SLD}$  приведены в табл. 4.1. Следует, в частности, отметить, что узел *Bucharest* впервые появляется в периферии на этапе, показанном на рис. 4.2,  $\delta$ , но не выбирается для развертывания, поскольку его  $f$ -стоимость (450) выше, чем стоимость узла *Pitesti* (417). Иными словами эту ситуацию можно описать так, что может существовать решение, при котором путь проходит через город Питешти со стоимостью, достигающей 417, поэтому алгоритм не останавливается на решении со стоимостью 450. Данный пример может служить общим свидетельством того, что ~~поиск A\* с использованием алгоритма Tree-Search является оптимальным, если функция h(n) допустима~~. Предположим, что на периферии поиска появился неоптимальный целевой узел  $G_2$ , а стоимость оптимального решения равна  $C^*$ . В таком случае, поскольку узел  $G_2$  неоптимален, а  $h(G_2) = 0$  (это выражение справедливо для любого целевого узла), можно вывести следующую формулу:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

Теперь рассмотрим периферийный узел  $n$ , который находится в оптимальном пути решения, например узел *Pitesti* в примере, приведенном в предыдущем абзаце. (Если решение существует, то всегда должен быть такой узел.) Если функция  $h(n)$  не переоценивает стоимость завершения этого пути решения, то справедлива следующая формула:

$$f(n) = g(n) + h(n) \leq C^*$$

Таким образом, доказано, что  $f(n) \leq C^* < f(G_2)$ , поэтому узел  $G_2$  не развертывается и поиск A\* должен вернуть оптимальное решение.

Если бы вместо алгоритма Tree-Search использовался алгоритм Graph-Search, приведенный в листинге 3.6, то данное доказательство стало бы недействительным. Дело в том, что алгоритм Graph-Search способен отбросить оптимальный путь к повторяющемуся состоянию, если он не был сформирован в первую очередь, поэтому может возвращать неоптимальные решения (см. упр. 4.4). Существуют два способа устранения этого недостатка. Первое решение состоит в том, что алгоритм Graph-Search должен быть дополнен так, чтобы он отбрасывал наиболее дорогостоящий из любых двух найденных путей к одному и тому же узлу (см. обсуждение этой темы в разделе 3.5). Сопровождение необходимой для этого дополнительной информации связано с определенными трудностями, но гарантирует оптимальность. Второе решение состоит в обеспечении того, чтобы оптимальный путь к любому повторяющемуся состоянию всегда был первым из тех, по которым следует алгоритм, как в случае поиска по критерию стоимости.

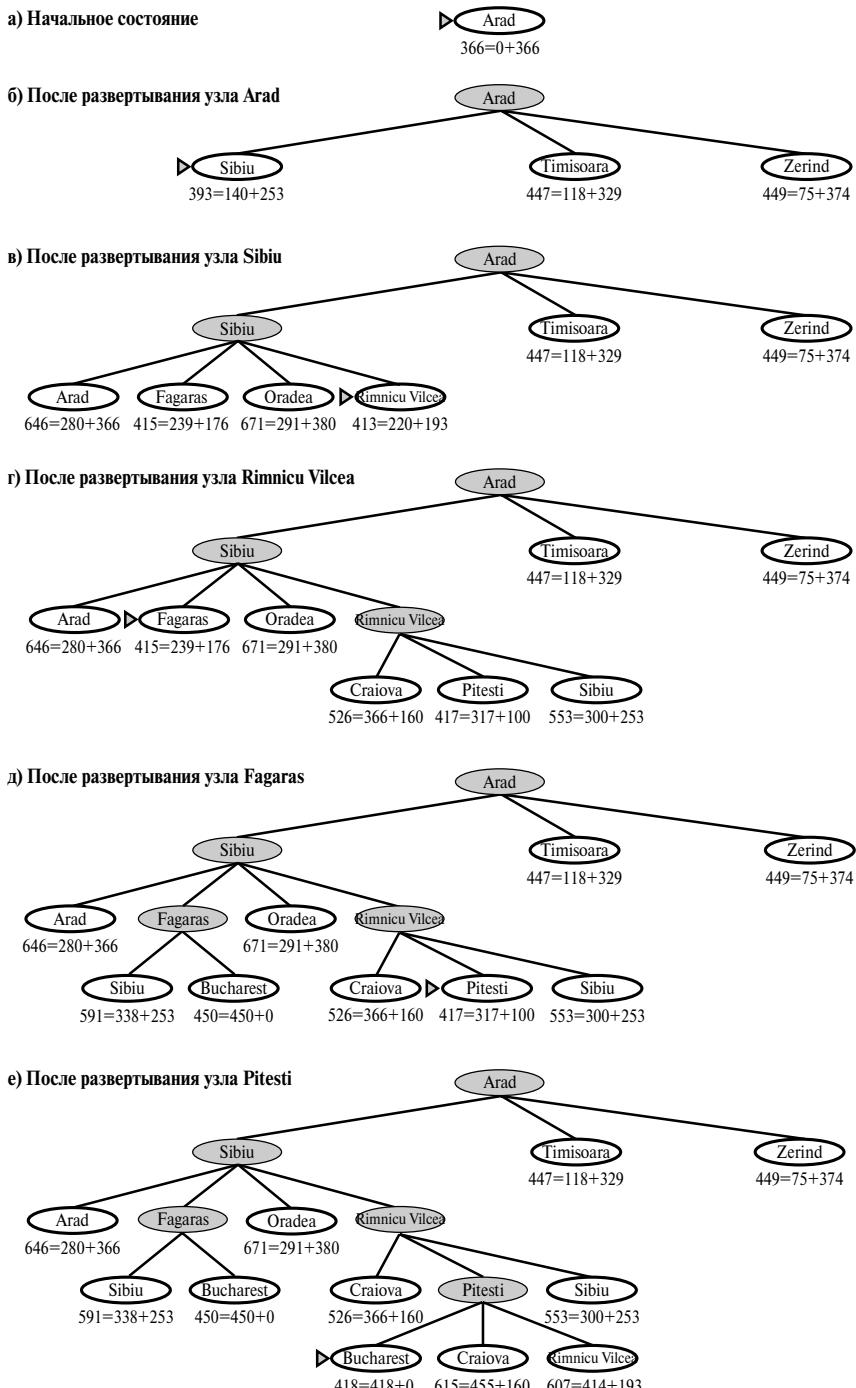


Рис. 4.2. Этапы поиска A\* пути в Бухарест. Узлы отмечены значениями  $f=g+h$ . Значения  $h$  представляют собой расстояния по прямой до Бухареста, приведенные в табл. 4.1

Такое свойство соблюдается, если на функцию  $h(n)$  налагается дополнительное требование, а именно требование обеспечения **преемственности** эвристической функции (такое свойство называют также **монотонностью** эвристической функции). Эвристическая функция  $h(n)$  является преемственной, если для любого узла  $n$  и для любого преемника  $n'$  узла  $n$ , сформированного в результате любого действия  $a$ , оценка стоимости достижения цели из узла  $n$  не больше чем стоимость этапа достижения узла  $n'$  плюс оценка стоимости достижения цели из узла  $n'$ :

$$h(n) \leq c(n, a, n') + h(n')$$

Это — форма общего **неравенства треугольника**, которое указывает, что длина любой стороны треугольника не может превышать сумму длин двух других сторон. В данном случае треугольник образован узлами  $n$ ,  $n'$  и целью, ближайшей к  $n$ . Можно довольно легко показать (упр. 4.7), что любая преемственная эвристическая функция является также допустимой. Наиболее важным следствием из определения преемственности является такой вывод: *если поиск A\* с использованием алгоритма Graph-Search является оптимальным, если функция  $h(n)$  преемственна.*

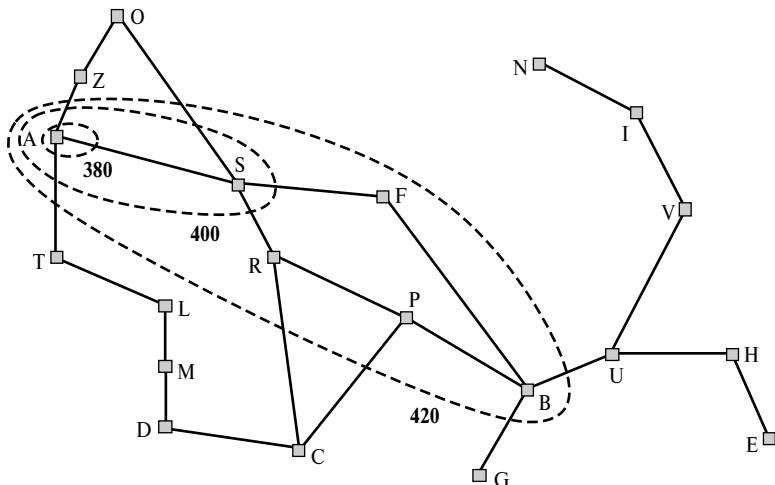
Несмотря на то что требование к преемственности является более строгим, чем требование к допустимости, весьма нелегко составить такие эвристические функции, которые были бы допустимыми, но не преемственными. Все допустимые эвристические функции, рассматриваемые в данной главе, являются также преемственными. Возьмем в качестве примера функцию  $h_{SLD}$ . Известно, что общее неравенство треугольника удовлетворяется, если длина каждой стороны измеряется с помощью расстояния по прямой, и что расстояние по прямой между  $n$  и  $n'$  не больше чем  $c(n, a, n')$ . Поэтому эвристическая функция  $h_{SLD}$  является преемственной.

Еще один важный вывод из определения преемственности является таковым: *если функция  $h(n)$  преемственна, то значения функции  $f(n)$  вдоль любого пути являются неубывающими*. Доказательство этого утверждения непосредственно вытекает из определения преемственности. Предположим, что узел  $n'$  — преемник узла  $n$ ; в таком случае для некоторого  $a$  справедливо выражение  $g(n') = g(n) + c(n, a, n')$  и имеет место такая формула:

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

На основании этого можно сделать вывод, что последовательность узлов, развернутых в поиске A\* с использованием алгоритма Graph-Search, находится в неубывающем порядке значений  $f(n)$ . Поэтому первый целевой узел, выбранный для развертывания, должен представлять собой оптимальное решение, поскольку все дальнейшие узлы будут, по меньшей мере, столь же дорогостоящими.

Тот факт, что  $f$ -стоимости вдоль любого пути являются неубывающими, означает также, что могут быть очерчены **контуры** равных  $f$ -стоимостей в пространстве состояний, полностью аналогичные контурам равных высот на топографической карте. Пример подобной схемы приведен на рис. 4.3. Внутри контура, обозначенного как 400, все узлы имеют значения  $f(n)$ , меньшие или равные 400, и т.д. В таком случае, поскольку в поиске A\* развертывается периферийный узел с наименьшей  $f$ -стоимостью, можно видеть, как поиск A\* распространяется из начального узла, добавляя узлы в виде концентрических полос с возрастающей  $f$ -стоимостью.



*Рис. 4.3. Карта Румынии, на которой показаны контуры, соответствующие  $f=380$ ,  $f=400$  и  $f=420$ , при том что Arad является начальным состоянием. Узлы в пределах данного конкретного контура имеют  $f$ -стоимости, меньшие или равные значению стоимости контура*

При поиске по критерию стоимости (таковым является поиск  $A^*$  с применением  $h(n)=0$ ) эти полосы будут представлять собой “кольца” с центром в начальном состоянии. При использовании более точных эвристических функций полосы вытягиваются в направлении целевого состояния и становятся более узко сосредоточенными вокруг оптимального пути. Если  $C^*$  представляет собой стоимость оптимального пути решения, то можно утверждать следующее:

- в поиске  $A^*$  развертываются все узлы со значениями  $f(n) < C^*$ ;
- поэтому в поиске  $A^*$  могут развертываться некоторые дополнительные узлы, находящиеся непосредственно на “целевом контуре” (где  $f(n) = C^*$ ), прежде чем будет выбран целевой узел.

На интуитивном уровне представляется очевидным, что первое найденное решение должно быть оптимальным, поскольку целевые узлы во всех последующих контурах будут иметь более высокое значение  $f$ -стоимости и поэтому более высокое значение  $g$ -стоимости (поскольку все целевые узлы имеют значения  $h(n)=0$ ). Кроме того, на интуитивном уровне также очевидно, что поиск  $A^*$  является полным. По мере добавления полос с возрастающими значениями  $f$  мы должны в конечном итоге достичь полосы, в которой значение  $f$  будет равно стоимости пути к целевому состоянию<sup>4</sup>.

Следует отметить, что в поиске  $A^*$  узлы со значением  $f(n) > C^*$  не развертываются; например, как показано на рис. 4.2, не развертывается узел *Timisoara*, даже несмотря на то, что является дочерним узлом корневого узла. Эту ситуацию принято обозначать так, что происходит **отсечение** поддерева, находящегося ниже узла

<sup>4</sup> Для соблюдения требования полноты необходимо, чтобы количество узлов со стоимостью, меньшей или равной  $C^*$ , было конечным; это условие соблюдается, если стоимости всех этапов превышают некоторое конечное значение  $\epsilon$ , а коэффициент ветвления  $b$  является конечным.

*Timisoara*; поскольку функция  $h_{SLD}$  является допустимой, рассматриваемый алгоритм может безопасно игнорировать это поддерево, гарантируя вместе с тем оптимальность. Понятие *отсечения* (под которым подразумевается исключение из рассмотрения некоторых вариантов в связи с отсутствием необходимости их исследовать) является важным для многих областей искусственного интеллекта.

Одно заключительное наблюдение состоит в том, что среди оптимальных алгоритмов такого типа (алгоритмов, которые развертывают пути поиска от корня) поиск A\* является ~~о~~ оптимально эффективным для любой конкретной эвристической функции. Это означает, что не гарантируется развертывание меньшего количества узлов, чем в поиске A\*, с помощью какого-либо иного оптимального алгоритма (не считая той возможности, когда осуществляется выбор на равных среди узлов с  $f(n) = C^*$ ). Это связано с тем, что любой алгоритм, который не развертывает все узлы со значениями  $f(n) < C^*$ , подвержен риску потери оптимального решения.

Те соображения, что поиск A\*, как один из всех подобных алгоритмов, является действительно полным, оптимальным и оптимально эффективным, оставляют довольно приятное впечатление. Но, к сожалению, это отнюдь не означает, что поиск A\* может служить ответом на все наши потребности в поиске. Сложность заключается в том, что при решении большинства задач количество узлов в пределах целевого контура пространства состояний все еще зависит экспоненциально от длины решения. Хотя доказательство этого утверждения выходит за рамки настоящей книги, было показано, что экспоненциальный рост происходит, если ошибка эвристической функции растет не быстрее по сравнению с логарифмом фактической стоимости пути. В математических обозначениях условие субэкспоненциального роста состоит в следующем:

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

где  $h^*(n)$  — истинная стоимость достижения цели из узла  $n$ . Почти для всех практически применяемых эвристических функций эта ошибка, по меньшей мере, пропорциональна стоимости пути, и происходящий в связи с этим экспоненциальный рост в конечном итоге превосходит возможности любого компьютера. По этой причине на практике стремление находить оптимальное решение часто не оправдано. Иногда вместо этого целесообразно использовать варианты поиска A\*, позволяющие быстро находить неоптимальные решения, а в других случаях — разрабатывать эвристические функции, которые являются более точными, но не строго допустимыми. В любом случае применение хорошей эвристической функции все равно обеспечивает поразительную экономию усилий по сравнению с использованием неинформированного поиска. Вопрос разработки хороших эвристических функций рассматривается в разделе 4.2.

Но большая продолжительность вычислений не является основным недостатком поиска A\*. Поскольку при поиске A\* все сформированные узлы хранятся в памяти (как и во всех алгоритмах Graph-Search), фактически ресурсы пространства исчерпываются задолго до того, как исчерпываются ресурсы времени. По этой причине поиск A\* не является практически применимым при решении многих крупномасштабных задач. Разработанные недавно алгоритмы позволяют преодолеть эту проблему пространства, не жертвуя оптимальностью или полнотой, за счет небольшого увеличения времени выполнения. Эти алгоритмы рассматриваются ниже.

## Эвристический поиск с ограничением объема памяти

Простейший способ сокращения потребностей в памяти для поиска A\* состоит в применении идеи итеративного углубления в контексте эвристического поиска. Реализация этой идеи привела к созданию алгоритма A\* с итеративным углублением (Iterative-Deepening A\* — IDA\*). Основное различие между алгоритмом IDA\* и стандартным алгоритмом итеративного углубления состоит в том, что применяемым условием останова развертывания служит f-стоимость ( $g+h$ ), а не глубина; на каждой итерации этим остановочным значением является минимальная f-стоимость любого узла, превышающая остановочное значение, достигнутое в предыдущей итерации. Алгоритм IDA\* является практически применимым для решения многих задач с единичными стоимостями этапов и позволяет избежать существенных издержек, связанных с поддержкой отсортированной очереди узлов. К сожалению, этот алгоритм характеризуется такими же сложностями, связанными с использованием стоимостей с действительными значениями, как и итеративная версия поиска по критерию стоимости, которая описана в упр. 3.11. В данном разделе кратко рассматриваются два более современных алгоритма с ограничением памяти, получивших названия RBFS и MA\*.

❖ **Рекурсивный поиск по первому наилучшему совпадению** (Recursive Best-First Search — RBFS) — это простой рекурсивный алгоритм, в котором предпринимаются попытки имитировать работу стандартного поиска по первому наилучшему совпадению, но с использованием только линейного пространства. Этот алгоритм приведен в листинге 4.1. Он имеет структуру, аналогичную структуре рекурсивного поиска в глубину, но вместо бесконечного следования вниз по текущему пути данный алгоритм контролирует f-значение наилучшего альтернативного пути, доступного из любого предка текущего узла. Если текущий узел превышает данный предел, то текущий этап рекурсии отменяется и рекурсия продолжается с альтернативного пути. После отмены данного этапа рекурсии в алгоритме RBFS происходит замена f-значения каждого узла вдоль данного пути наилучшим f-значением его дочернего узла. Благодаря этому в алгоритме RBFS запоминается f-значение наилучшего листового узла из забытого поддерева и поэтому в некоторый последующий момент времени может быть принято решение о том, стоит ли снова развертывать это поддерево. На рис. 4.4 показано, как с помощью алгоритма RBFS происходит поиск пути в Бухарест.

### Листинг 4.1. Алгоритм рекурсивного поиска по первому наилучшему совпадению

```
function Recursive-Best-First-Search(problem) returns решение result
или индикатор неудачи failure
    RBFS(problem, Make-Node(Initial-State[problem]),  $\infty$ )
```

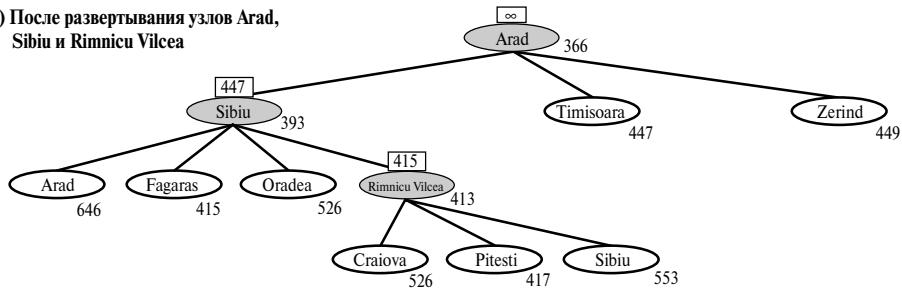
```
function RBFS(problem, node, f_limit) returns решение result
или индикатор неудачи failure и новый предел f-стоимости f_limit
    if Goal-Test[problem](State[node]) then return узел node
    successors  $\leftarrow$  Expand(node, problem)
    if множество узлов-преемников successors пусто
        then return failure,  $\infty$ 
    for each s in successors do
        f[s]  $\leftarrow$  max(g(s)+h(s), f[node])
    repeat
        best  $\leftarrow$  узел с наименьшим f-значением в множестве successors
        if f[best] > f_limit then return failure, f[best]
```

```

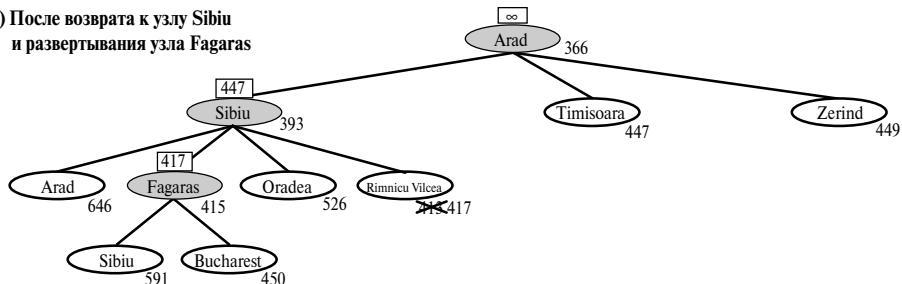
alternative ← второе после наименьшего f-значение
в множестве successors
result, f[best] ← RBFS(problem, best,
min(f_limit, alternative))
if result ≠ failure then return result

```

а) После развертывания узлов Arad, Sibiu и Rimnicu Vilcea



б) После возврата к узлу Sibiu и развертывания узла Fagaras



в) После переключения снова на узел Rimnicu Vilcea и развертывания узла Pitesti

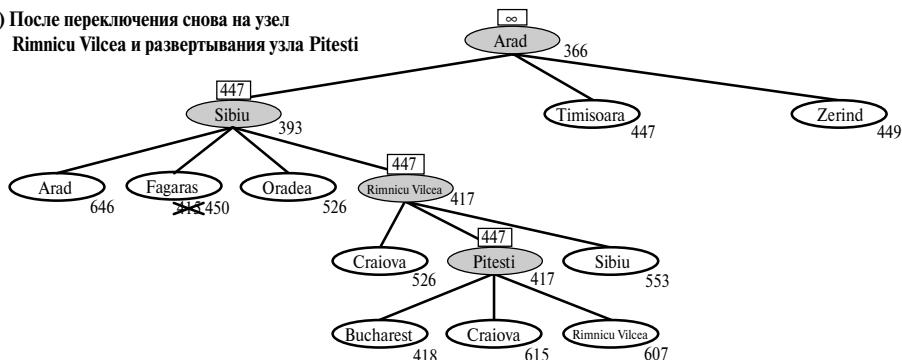


Рис. 4.4. Этапы поиска кратчайшего маршрута в Бухарест с помощью алгоритма RBFS. Значение f-предела для каждого рекурсивного вызова показано над каждым текущим узлом: путь через узел Rimnicu Vilcea, который следует до текущего наилучшего листового узла (Pitesti), имеет значение, худшее по сравнению с нашумшим альтернативным путем (Fagaras) (а); рекурсия продолжается и значение наилучшего листового узла забытого поддерева (417) резервируется в узле Rimnicu Vilcea; затем развертывается узел Fagaras, в результате чего обнаруживается наилучшее значение листового узла, равное 450 (б); рекурсия продолжается и значение наилучшего листового узла забытого поддерева (450) резервируется в узле Fagaras; затем развертывается узел Rimnicu Vilcea; на этом раз развертывание продолжается в сторону Бухареста, поскольку наилучший альтернативный путь (через узел Timisoara) стоит, по меньшей мере, 447 (в)

Алгоритм RBFS является немного более эффективным по сравнению с IDA\*, но все еще страдает от недостатка, связанного со слишком частым повторным формированием узлов. В примере, приведенном на рис. 4.4, алгоритм RBFS вначале следует по пути через узел *Rimnicu Vilcea*, затем “меняет решение” и пытается пройти через узел *Fagaras*, после этого снова возвращается к отвергнутому ранее решению. Такие смены решения происходят в связи с тем, что при каждом развертывании текущего наилучшего пути велика вероятность того, что его f-значение увеличится, поскольку функция  $h$  обычно становится менее оптимистической по мере того, как происходит развертывание узлов, более близких к цели. После того как возникает такая ситуация, особенно в больших пространствах поиска, путь, который был вторым после наилучшего, может сам стать наилучшим путем, поэтому в алгоритме поиска приходится выполнять возврат, чтобы проследовать по нему. Каждое изменение решения соответствует одной итерации алгоритма IDA\* и может потребовать многих повторных развертываний забытых узлов для воссоздания наилучшего пути и развертывания пути еще на один узел.

Как и алгоритм поиска A\*, RBFS является оптимальным алгоритмом, если эвристическая функция  $h(n)$  допустима. Его пространственная сложность равна  $O(bd)$ , но охарактеризовать временную сложность довольно трудно: она зависит и от точности эвристической функции, и от того, насколько часто происходила смена наилучшего пути по мере развертывания узлов. И алгоритм IDA\*, и алгоритм RBFS подвержены потенциальному экспоненциальному увеличению сложности, связанной с поиском в графах (см. раздел 3.5), поскольку эти алгоритмы не позволяют определять наличие повторяющихся состояний, отличных от тех, которые находятся в текущем пути. Поэтому данные алгоритмы способны много раз исследовать одно и то же состояние.

Алгоритмы IDA\* и RBFS страдают от того недостатка, что в них используется слишком мало памяти. Между итерациями алгоритм IDA\* сохраняет только единственное число — текущий предел f-стоимости. Алгоритм RBFS сохраняет в памяти больше информации, но количество используемой в нем памяти измеряется лишь значением  $O(bd)$ : даже если бы было доступно больше памяти, алгоритм RBFS не способен ею воспользоваться.

Поэтому представляется более разумным использование всей доступной памяти. Двумя алгоритмами, которые осуществляют это требование, являются поиск  $\Delta$  MA\* (Memory-bounded A\* — поиск A\* с ограничением памяти) и  $\Delta$  SMA\* (Simplified MA\* — упрощенный поиск MA\*). В данном разделе будет описан алгоритм SMA\*, который действительно является более простым, чем другие алгоритмы этого типа. Алгоритм SMA\* действует полностью аналогично поиску A\*, развертывая наилучшие листовые узлы до тех пока, пока не будет исчерпана доступная память. С этого момента он не может добавить новый узел к дереву поиска, не уничтожив старый. В алгоритме SMA\* всегда уничтожается наихудший листовой узел (тот, который имеет наибольшее f-значение). Как и в алгоритме RBFS, после этого в алгоритме SMA\* значение забытого (уничтоженного) узла резервируется в его родительском узле. Благодаря этому предок забытого поддерева позволяет определить качество наилучшего пути в этом поддереве. Поскольку имеется данная информация, в алгоритме SMA\* поддерево восстанавливается, только если обнаруживается, что все другие пути выглядят менее многообещающими по сравнению с забытым путем. Иными словами, если все потомки узла  $n$  забыты, то неизвестно, каким

путем можно следовать от  $n$ , но все еще можно получить представление о том, есть ли смысл куда-либо следовать от  $n$ .

Полный алгоритм слишком сложен для того, чтобы его можно было воспроизвести в данной книге<sup>5</sup>, но заслуживает упоминания один его нюанс. Как уже было сказано выше, в алгоритме SMA\* развертывается наилучший листовой узел и удаляется наихудший листовой узел. А что происходит, если все листовые узлы имеют одинаковое  $f$ -значение? В таком случае может оказаться, что алгоритм выбирает для удаления и развертывания один и тот же узел. В алгоритме SMA\* эта проблема решается путем развертывания самого нового наилучшего листового узла и удаления самого старого наихудшего листового узла. Эти два узла могут оказаться одним и тем же узлом, только если существует лишь один листовой узел; в таком случае текущее дерево поиска должно представлять собой единственный путь от корня до листового узла, заполняющий всю память. Это означает, что если данный листовой узел не является целевым узлом, то решение не достижимо при доступном объеме памяти, даже если этот узел находится в оптимальном пути решения. Поэтому такой узел может быть отброшен точно так же, как и в том случае, если он не имеет преемников.

Алгоритм SMA\* является полным, если существует какое-либо достижимое решение, иными словами, если  $d$ , глубина самого поверхностного целевого узла, меньше чем объем памяти (выраженный в хранимых узлах). Этот алгоритм оптимален, если достижимо какое-либо оптимальное решение; в противном случае он возвращает наилучшее достижимое решение. С точки зрения практики алгоритм SMA\* вполне может оказаться наилучшим алгоритмом общего назначения для поиска оптимальных решений, особенно если пространство состояний представляет собой граф, стоимости этапов не одинаковы, а операция формирования узлов является более дорогостоящей в сравнении с дополнительными издержками сопровождения открытых и закрытых списков.

Однако при решении очень сложных задач часто возникают ситуации, в которых алгоритм SMA\* вынужден постоянно переключаться с одного пути решения на другой в пределах множества возможных путей решения, притом что в памяти может поместиться только небольшое подмножество этого множества. (Такие ситуации напоминают проблему  **пробуксовки** в системах подкачки страниц с жесткого диска.) В таком случае на повторное формирование одних и тех узлов затрачивается дополнительное время, а это означает, что задачи, которые были бы фактически разрешимыми с помощью поиска A\* при наличии неограниченной памяти, становятся трудноразрешимыми для алгоритма SMA\*. Иными словами, из-за ~~ограничений~~ ограничений в объеме памяти некоторые задачи могут становиться трудноразрешимыми с точки зрения времени вычисления. Хотя отсутствует теория, позволяющая найти компромисс между затратами времени и памяти, создается впечатление, что зачастую избежать возникновения этой проблемы невозможно. Единственным способом преодоления такой ситуации становится частичный отказ от требований к оптимальности решения.

## Обучение лучшим способам поиска

Выше было представлено несколько стратегий поиска (поиск в ширину, жадный поиск по первому наилучшему совпадению и т.д.), которые были разработаны уч-

---

<sup>5</sup> Грубый набросок этого алгоритма был приведен в первом издании настоящей книги.

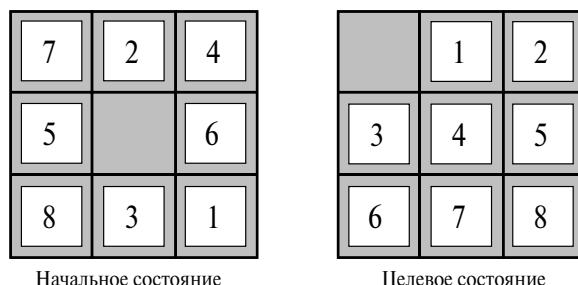
ными и специалистами по компьютерным наукам. Но может ли сам агент обучаться лучшим способам поиска? Ответ на этот вопрос является положительным, а применяемый при этом метод обучения опирается на важную концепцию, называемую *пространством состояний, рассматриваемым на метауровне*, или **метауровневым пространством состояний**. Каждое состояние в метауровневом пространстве состояний отражает внутреннее (вычислительное) состояние программы, выполняющей поиск в пространстве состояний, рассматриваемом на уровне объектов, или в **объектно-уровневом пространстве состояний**, таком как карта Румынии. Например, внутреннее состояние алгоритма A\* включает в себя текущее дерево поиска. Каждое действие в метауровневом пространстве состояний представляет собой этап вычисления, который изменяет внутреннее состояние, например, на каждом этапе вычисления в процессе поиска A\* развертывается один из листовых узлов, а его преемники добавляются к дереву. Таким образом, рис. 4.2, на котором показана последовательность всех больших и больших деревьев поиска, может рассматриваться как изображающий путь в метауровневом пространстве состояний, где каждое состояние в пути является объектно-уровневым деревом поиска.

В настоящее время путь, показанный на рис. 4.2, имеет пять этапов, включая один этап (развертывание узла *Fagaras*), который нельзя назвать слишком полезным. Может оказаться, что при решении более сложных задач количество подобных ненужных этапов будет намного больше, а алгоритм **метауровневого обучения** может изучать этот опыт, чтобы в дальнейшем избегать исследования бесперспективных поддеревьев. Методы, используемые при обучении такого рода, описаны в главе 21. Целью обучения является минимизация **суммарной стоимости** решения задач, а также поиск компромисса между вычислительными издержками и стоимостью пути.

## 4.2. ЭВРИСТИЧЕСКИЕ ФУНКЦИИ

В этом разделе будут рассматриваться эвристические функции для задачи игры в восемь, что позволяет лучше продемонстрировать характерные особенности всех эвристических функций в целом.

Головоломка “игра в восемь” была одной из первых задач эвристического поиска. Как было указано в разделе 3.2, в ходе решения этой головоломки требуется передвигать фишку по горизонтали или по вертикали на пустой участок до тех пор, пока полученная конфигурация не будет соответствовать целевой конфигурации (рис. 4.5).



*Рис. 4.5. Типичный экземпляр головоломки “игра в восемь”. Решение имеет длину 26 этапов*

Средняя стоимость решения для сформированного случайным образом экземпляра головоломки “игра в восемь” составляет около 22 этапов. Коэффициент ветвления примерно равен 3. (Если пустой квадрат находится в середине коробки, то количество возможных ходов равно четырем, если находится в углу — двум, а если в середине одной из сторон — трем.) Это означает, что при исчерпывающем поиске на глубину 22 приходится рассматривать примерно  $3^{22} \approx 3 \cdot 1 \times 10^{10}$  состояний. Отслеживая повторяющиеся состояния, это количество состояний можно сократить приблизительно в 170 000 раз, поскольку существует только  $9! / 2 = 181 \cdot 440$  различных состояний, которые являются достижимыми (см. упр. 3.4.) Это количество состояний уже лучше поддается контролю, но соответствующее количество для игры в пятнадцать примерно равно  $10^{13}$ , поэтому для такой головоломки с более высокой сложностью требуется найти хорошую эвристическую функцию. Если нужно находить кратчайшие решения с использованием поиска A\*, то требуется эвристическая функция, которая никогда не переоценивает количество этапов достижения цели. История исследований в области поиска таких эвристических функций для игры в пятнадцать является довольно долгой, а в данном разделе рассматриваются два широко используемых кандидата на эту роль, которые описаны ниже.

- $h_1$  = количество фишек, стоящих не на своем месте. На рис. 4.5 все восемь фишек стоят не на своем месте, поэтому показанное слева начальное состояние имеет эвристическую оценку  $h_1=8$ . Эвристическая функция  $h_1$  является допустимой, поскольку очевидно, что каждую фишку, находящуюся не на своем месте, необходимо переместить по меньшей мере один раз.
- $h_2$  = сумма расстояний всех фишек от их целевых позиций. Поскольку фишки не могут передвигаться по диагонали, рассчитываемое расстояние представляет собой сумму горизонтальных и вертикальных расстояний. Такое расстояние иногда называют **расстоянием, измеряемым в городских кварталах**, или **манхэттенским расстоянием**. Эвристическая функция  $h_2$  также является допустимой, поскольку все, что может быть сделано в одном ходе, состоит лишь в перемещении одной фишки на один этап ближе к цели. Фишки от 1 до 8 в рассматриваемом начальном состоянии соответствуют такому значению манхэттенского расстояния:  $h_2=3+1+2+2+2+3+3+2=18$ .

Как и можно было предположить, ни одна из этих функций не переоценивает истинную стоимость решения, которая равна 26.

### Зависимость производительности поиска от точности эвристической функции

Одним из критериев, позволяющих охарактеризовать качество эвристической функции, является **эффективный коэффициент ветвления**  $b^*$ . Если общее количество узлов, вырабатываемых в процессе поиска A\* решения конкретной задачи, равно  $N$ , а глубина решения равна  $d$ , то  $b^*$  представляет собой коэффициент ветвления, который должно иметь однородное дерево с глубиной  $d$  для того, чтобы в нем содержалось  $N+1$  узлов. Поэтому справедлива следующая формула:

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Например, если алгоритм A\* находит решение на глубине 5 с использованием 52 узлов, то эффективный коэффициент ветвления равен 1,92. Эффективный коэффи-

циент ветвления может изменяться от одного экземпляра одной и той же задачи к другому, но обычно в случае достаточно трудных задач остается относительно постоянным. Поэтому экспериментальные измерения коэффициента  $b^*$  на небольшом множестве задач могут служить хорошим критерием общей полезности рассматриваемой эвристической функции. Хорошо спроектированная эвристическая функция должна иметь значение  $b^*$ , близкое к 1, что позволяет быстро решать довольно большие задачи.

Для проверки эвристических функций  $h_1$  и  $h_2$  авторы сформировали случайным образом 1200 экземпляров задачи с длиной решения от 2 до 24 (по 100 экземпляров для каждого четного значения длины) и нашли их решения с помощью поиска с итеративным углублением и поиска в дереве по алгоритму A\* с применением эвристических функций  $h_1$  и  $h_2$ . Данные о среднем количестве узлов, развернутых при использовании каждой стратегии и эффективном коэффициенте ветвления, приведены в табл. 4.2. Эти результаты показывают, что эвристическая функция  $h_2$  лучше чем  $h_1$  и намного лучше по сравнению с использованием поиска с итеративным углублением. Применительно к найденным авторами решениям с длиной 14 применение поиска A\* с эвристической функцией  $h_2$  становится в 30 000 раз более эффективным по сравнению с неинформированным поиском с итеративным углублением.

**Таблица 4.2. Сравнение значений стоимости поиска и эффективного коэффициента ветвления для алгоритмов Iterative-Deepening-Search и A\* с  $h_1$ ,  $h_2$ . Данные усреднялись по 100 экземплярам задачи игры в восемь применительно к различным значениям длины решения**

$d$	IDS	Стоимость поиска		Эффективный коэффициент ветвления	
		A*( $h_1$ )	A*( $h_2$ )	IDS	A*( $h_1$ )
2	10	6	6	2,45	1,79
4	112	13	12	2,87	1,48
6	680	20	18	2,73	1,34
8	6384	39	25	2,80	1,33
10	47127	93	39	2,79	1,38
12	3644035	227	73	2,78	1,42
14	-	539	113	-	1,44
16	-	1301	211	-	1,45
18	-	3056	363	-	1,46
20	-	7276	676	-	1,47
22	-	18094	1219	-	1,48
24	-	39135	1641	-	1,48

Интерес представляет вопрос о том, всегда ли эвристическая функция  $h_2$  лучше чем  $h_1$ . Ответ на этот вопрос является положительным. На основании определений этих двух эвристических функций можно легко прийти к выводу, что для любого узла  $n$  справедливо выражение  $h_2(n) \geq h_1(n)$ . Таким образом, можно утверждать, что эвристика  $h_2$  ~~доминирует~~ над  $h_1$ . Доминирование напрямую связано с эффективностью: при поиске A\* с использованием функции  $h_2$  никогда не происходит развертывание большего количества узлов, чем при поиске A\* с использованием  $h_1$  (возможно, за исключением нескольких узлов с  $f(n) = C^*$ ). Доказательство этого ут-

верждения является несложным. Напомним приведенное на с. 161 замечание, что каждый узел со значением  $f(n) < C^*$  наверняка должен быть развернут. Это аналогично утверждению, что должен быть наверняка развернут каждый узел со значением  $h(n) < C^* - g(n)$ . Но поскольку для всех узлов значение  $h_2$ , по крайней мере, не меньше значения  $h_1$ , то каждый узел, который должен быть наверняка развернут в поиске  $A^*$  с  $h_2$ , будет также наверняка развернут при поиске с  $h_1$ , а применение эвристической функции  $h_1$  может к тому же вызвать и развертывание других узлов. Поэтому всегда лучше использовать эвристическую функцию с более высокими значениями, при тех условиях, что эта функция не переоценивает длину пути решения и что время вычисления этой эвристической функции не слишком велико.

### Составление допустимых эвристических функций

Выше было показано, что эвристические функции  $h_1$  (в которой используется количество стоящих не на своем месте фишек) и  $h_2$  (в которой используется манхэттенское расстояние) являются довольно неплохими эвристическими функциями для задачи игры в восемь и что функция  $h_2$  — из них наилучшая. Но на основании чего именно была предложена функция  $h_2$ ? Возможно ли, чтобы компьютер мог составить некоторую эвристическую функцию механически?

Эвристические функции  $h_1$  и  $h_2$  представляют собой оценки оставшейся длины пути для задачи игры в восемь, но они, кроме того, возвращают идеально точные значения длины пути для упрощенных версий этой игры. Если бы правила игры в восемь изменились таким образом, чтобы любую фишку можно было передвигать куда угодно, а не только на соседний пустой квадрат, то эвристическая функция  $h_1$  возвращала бы точное количество этапов в кратчайшем решении. Аналогичным образом, если бы любую фишку можно было перемещать на один квадрат в любом направлении, даже на занятый квадрат, то точное количество этапов в кратчайшем решении возвращала бы эвристическая функция  $h_2$ . Задача с меньшим количеством ограничений на возможные действия называется **ослабленной задачей**. Стоимость оптимального решения ослабленной задачи представляет собой допустимую эвристику для первоначальной задачи. Такая эвристическая функция является допустимой, поскольку оптимальное решение первоначальной задачи, по определению, служит также решением ослабленной задачи и поэтому должно быть, по меньшей мере, таким же дорогостоящим, как и оптимальное решение ослабленной задачи. Поскольку значение производной эвристической функции представляет собой точную стоимость решения ослабленной задачи, эта функция должно подчиняться неравенству треугольника и поэтому должна быть **преемственной** (см. с. 160).

Если определение задачи записано на каком-то формальном языке, то существует возможность формировать ослабленные задачи автоматически<sup>6</sup>. Например, если действия в игре в восемь описаны следующим образом:

---

<sup>6</sup> В главах 8 и 11 будут описаны формальные языки, подходящие для этого назначения; возможность автоматизировать формирование ослабленных задач появляется при наличии формальных описаний, с которыми могут проводиться манипуляции, а пока для этого назначения будет использоваться естественный язык.

Фишка может быть передвинута из квадрата  $A$  в квадрат  $B$ , если квадрат  $A$  является смежным по горизонтали или по вертикали с квадратом  $B$  и квадрат  $B$  пуст

то могут быть сформированы три ослабленные задачи путем удаления одного или обоих из приведенных выше условий.

- Фишка может быть передвинута из квадрата  $A$  в квадрат  $B$ , если квадрат  $A$  является смежным с квадратом  $B$ .
- Фишка может быть передвинута из квадрата  $A$  в квадрат  $B$ , если квадрат  $B$  пуст.
- Фишка может быть передвинута из квадрата  $A$  в квадрат  $B$ .

На основании ослабленной задачи а) можно вывести функцию  $h_2$  (манхэттенское расстояние). В основе этих рассуждений лежит то, что  $h_2$  должна представлять собой правильную оценку, если каждая фишка передвигается к месту ее назначения по очереди. Эвристическая функция, полученная на основании ослабленной задачи б), обсуждается в упр. 4.9. На основании ослабленной задачи в) можно получить эвристическую функцию  $h_1$  (фишки, стоящие не на своих местах), поскольку эта оценка была бы правильной, если бы фишки можно было передвигать в предназначеннное для них место за один этап. Обратите внимание на то, что здесь существенной является возможность решать ослабленные задачи, создаваемые с помощью указанного метода, фактически без какого-либо поиска, поскольку ослабленные правила обеспечивают декомпозицию этой задачи на восемь независимых подзадач. Если бы было трудно решать и ослабленную задачу, то был бы дорогостоящим сам процесс получения значений соответствующей эвристической функции<sup>7</sup>.

Для автоматического формирования эвристических функций на основе определений задач с использованием метода “ослабленной задачи” и некоторых других методов может применяться программа под названием Absolver [1237]. Программа Absolver составила для задачи игры в восемь новую эвристическую функцию, лучшую по сравнению со всеми существовавшими ранее эвристическими функциями, а также нашла первую полезную эвристическую функцию для знаменитой головоломки “кубик Рубика”.

Одна из проблем, возникающих при составлении новых эвристических функций, состоит в том, что часто не удается получить эвристическую функцию, которая была бы “лучшей во всех отношениях” по сравнению с другими. Если для решения какой-либо задачи может применяться целая коллекция допустимых эвристических функций  $h_1 \dots h_m$  и ни одна из них не доминирует над какой-либо из других функций, то какая из этих функций должна быть выбрана? Оказалось, что такой выбор делать не требуется, поскольку можно взять от них всех самое лучшее, определив такой критерий выбора:

$$h(n) = \max\{h_1(n), \dots, h_m(n)\}$$

В этой составной эвристике используется та функция, которая является наиболее точной для рассматриваемого узла. Поскольку эвристические функции, входящие в состав эвристики  $h$ , являются допустимыми, сама эта функция также является до-

<sup>7</sup> Следует отметить, что идеальную эвристическую функцию можно также получить, просто разрешив функции  $h$  “тайком” выполнять полный поиск в ширину. Таким образом, при создании эвристических функций всегда приходится искать компромисс между точностью и временем вычисления.

пустимой; кроме того, можно легко доказать, что функция  $h$  преемственна. К тому же  $h$  доминирует над всеми эвристическими функциями, которые входят в ее состав.

Допустимые эвристические функции могут быть также выведены на основе стоимости решения  $\bowtie$  подзадачи данной конкретной задачи. Например, на рис. 4.6 показана подзадача для экземпляра игры в восемь, приведенного на рис. 4.5. Эта подзадача касается перемещения фишек 1, 2, 3, 4 в их правильные позиции. Очевидно, что стоимость оптимального решения этой подзадачи представляет собой нижнюю границу стоимости решения полной задачи. Как оказалось, такая оценка в некоторых случаях является намного более точной по сравнению с манхэттенским расстоянием.

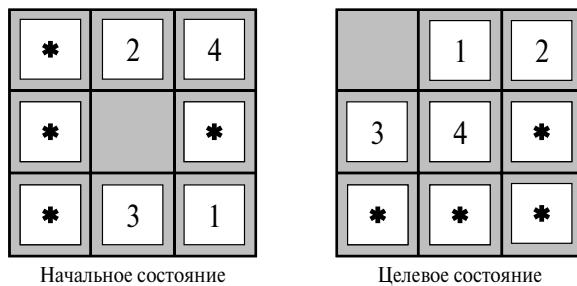


Рис. 4.6. Подзадача для экземпляра игры в восемь, показанного на рис. 4.5. Задание заключается в том, чтобы передвинуть фишки 1, 2, 3 и 4 в их правильные позиции, не беспокоясь о том, что произойдет с другими фишками

В основе  $\bowtie$  баз данных с шаблонами лежит такая идея, что указанные точные стоимости решений нужно хранить для каждого возможного экземпляра подзадачи — в нашем примере для каждой возможной конфигурации из четырех фишек и пустого квадрата. (Следует отметить, что при выполнении задания по решению этой подзадачи местонахождения остальных четырех фишек не нужно принимать во внимание, но ходы с этими фишками следует учитывать в стоимости решения.) После этого вычисляется допустимая эвристическая функция  $h_{DB}$  (здесь DB — Data Base) для каждого полного состояния, встретившегося в процессе поиска, путем выборки данных для соответствующей конфигурации подзадачи из базы данных. Сама база данных заполняется путем обратного поиска из целевого состояния и регистрации стоимости каждого нового встретившегося шаблона; затраты на этот поиск окупаются за счет успешного решения в дальнейшем многих новых экземпляров задачи.

Выбор фишек 1–2–3–4 является довольно произвольным; можно было бы также создать базы данных для фишек 5–6–7–8, 2–4–6–8 и т.д. По данным каждой базы данных формируется допустимая эвристическая функция, а эти эвристические функции можно составлять в общую эвристическую функцию, как было описано выше, применяя их максимальное значение. Составная эвристическая функция такого вида является намного более точной по сравнению с манхэттенским расстоянием; количество узлов, вырабатываемых при решении сформированных случайным образом экземпляров задачи игры в пятнадцать, может быть уменьшено примерно в 1000 раз.

Представляет интерес вопрос о том, можно ли складывать значения эвристических функций, полученных из баз данных 1–2–3–4 и 5–6–7–8, поскольку очевид-

но, что соответствующие две подзадачи не перекрываются. Будет ли при этом все еще получена допустимая эвристическая функция? Ответ на этот вопрос является отрицательным, поскольку в решениях подзадачи 1–2–3–4 и подзадачи 5–6–7–8 для данного конкретного состояния должны наверняка присутствовать некоторые общие ходы. Дело в том, что маловероятна ситуация, при которой фишк 1–2–3–4 можно было бы передвинуть на свои места, не трогая фишк 5–6–7–8, и наоборот. Но что будет, если не учитывать эти ходы? Иными словами, допустим, что регистрируется не общая стоимость решения подзадачи 1–2–3–4, а только количество ходов, в которых затрагиваются фишк 1–2–3–4. В таком случае можно легко определить, что сумма этих двух стоимостей все еще представляет собой нижнюю границу стоимости решения всей задачи. Именно эта идея лежит в основе **баз данных с непересекающимися шаблонами**. Применение таких баз данных позволяет решать случайно выбранные экземпляры задачи игры в пятнадцать за несколько миллисекунд — количество формируемых узлов сокращается примерно в 10 000 раз по сравнению с использованием манхэттенского расстояния. Для задачи игры в 24 может быть достигнуто ускорение приблизительно в миллион раз.

Базы данных с непересекающимися шаблонами успешно применяются для решения головоломок со скользящими фишками, поскольку сама задача может быть разделена таким образом, что каждый ход влияет лишь на одну подзадачу, так как одновременно происходит перемещение только одной фишк. При решении таких задач, как кубик Рубика, подобное разделение не может быть выполнено, поскольку каждый ход влияет на положение 8 или 9 из 26 элементов кубика. В настоящее время нет полного понимания того, как можно определить базы данных с непересекающимися шаблонами для подобных задач.

### Изучение эвристических функций на основе опыта

Предполагается, что эвристическая функция  $h(n)$  оценивает стоимость решения, начиная от состояния, связанного с узлом  $n$ . Как может некоторый агент составить подобную функцию? Одно из решений было приведено в предыдущем разделе, а именно: агент должен сформулировать ослабленные задачи, для которых может быть легко найдено оптимальное решение. Еще один подход состоит в том, что агент должен обучаться на основании полученного опыта. Здесь под “получением опыта” подразумевается, например, решение большого количества экземпляров игры в восемь. Каждое оптимальное решение задачи игры в восемь предоставляет примеры, на основе которых можно изучать функцию  $h(n)$ . Каждый пример складывается из состояния, взятого из пути решения, и фактической стоимости пути от этой точки до решения. На основе данных примеров с помощью алгоритма **индуктивного обучения** может быть сформирована некоторая функция  $h(n)$ , способная (при счастливом стечении обстоятельств) предсказывать стоимости решений для других состояний, которые возникают во время поиска. Методы осуществления именно такого подхода с использованием нейронных сетей, деревьев решений и других средств описаны в главе 18. (Применимы также методы обучения с подкреплением, которые рассматриваются в главе 21.)

Методы индуктивного обучения действуют лучше всего, когда в них учитываются **характеристики** состояния, релевантные для оценки этого состояния, а не просто общее описание состояния. Например, характеристика “количество стоящих не на

своих местах фишек” может быть полезной при предсказании фактического удаления некоторого состояния от цели. Назовем эту характеристику  $x_1(n)$ . Например, можно взять 100 сформированных случайным образом конфигураций головоломки игры в восемь и собрать статистические данные об их фактических стоимостях решений. Допустим, это позволяет обнаружить, что при  $x_1(n)$ , равном 5, средняя стоимость решения составляет около 14, и т.д. Наличие таких данных позволяет использовать значение  $x_1$  для предсказания значений функции  $h(n)$ . Безусловно, можно также применять сразу несколько характеристик. Второй характеристикой,  $x_2(n)$ , может быть “количество пар смежных фишек, которые являются также смежными в целевом состоянии”. Каким образом можно скомбинировать значения  $x_1(n)$  и  $x_2(n)$  для предсказания значения  $h(n)$ ? Общепринятый подход состоит в использовании линейной комбинации, как показано ниже.

$$h(n) = c_1 x_1(n) + c_2 x_2(n)$$

Константы  $c_1$  и  $c_2$  корректируются для достижения наилучшего соответствия фактическим данным о стоимостях решений. Предполагается, что константа  $c_1$  должна быть положительной, а  $c_2$  — отрицательной.

### **4.3. АЛГОРИТМЫ ЛОКАЛЬНОГО ПОИСКА И ЗАДАЧИ ОПТИМИЗАЦИИ**

---

Рассматривавшиеся до сих пор алгоритмы поиска предназначались для систематического исследования пространств поиска. Такая систематичность достигается благодаря тому, что один или несколько путей хранится в памяти и проводится регистрация того, какие альтернативы были исследованы в каждой точке вдоль этого пути, а какие нет. После того как цель найдена, путь к этой цели составляет также исключительное решение данной задачи.

Но при решении многих задач путь к цели не представляет интереса. Например, в задаче с восемью ферзями (см. с. 118) важна лишь окончательная конфигурация ферзей, а не порядок, в котором они были поставлены на доску. К этому классу задач относятся многие важные приложения, такие как проектирование интегральных схем, разработка плана цеха, составление производственного расписания, автоматическое программирование, оптимизация сети связи, составление маршрута транспортного средства и управление портфелем акций.

Если путь к цели не представляет интереса, то могут рассматриваться алгоритмы другого класса, в которых вообще не требуются какие-либо данные о путях. Алгоритмы **локального поиска** действуют с учетом единственного **текущего состояния** (а не многочисленных путей) и обычно предусматривают только переход в состояние, соседнее по отношению к текущему состоянию. Как правило, информация о путях, пройденных в процессе такого поиска, не сохраняется. Хотя алгоритмы локального поиска не предусматривают систематическое исследование пространства состояний (не являются систематическими), они обладают двумя важными преимуществами: во-первых, в них используется очень небольшой объем памяти, причем обычно постоянный, и, во-вторых, они часто позволяют находить приемлемые решения в больших или бесконечных (непрерывных) пространствах состояний, для которых систематические алгоритмы не применимы.

Кроме поиска целей, алгоритмы локального поиска являются полезным средством решения чистых задач оптимизации, назначение которых состоит в поиске состояния, наилучшего с точки зрения целевой функции. Многие задачи оптимизации не вписываются в “стандартную” модель поиска, представленную в главе 3. Например, природа предусмотрела такую целевую функцию (пригодность для репродукции), что дарвиновская эволюция может рассматриваться как попытка ее оптимизации, но в этой задаче оптимизации нет компонентов “проверка цели” и “стоимость пути”.

Авторы пришли к выводу, что для понимания сути локального поиска очень полезно рассмотреть ландшафт пространства состояний (подобный показанному на рис. 4.7). Этот ландшафт характеризуется и “местонахождением” (которое определяется состоянием), и “возвышением” (определенным значением эвристической функции стоимости или целевой функции). Если возвышение соответствует стоимости, то задача состоит в поиске самой глубокой долины — глобального минимума, а если возвышение соответствует целевой функции, то задача заключается в поиске высочайшего пика — глобального максимума. (Минимум и максимум можно поменять местами, взяв их с обратными знаками.) Алгоритмы локального поиска исследуют такой ландшафт. Алгоритм полного локального поиска всегда находит цель, если она существует, а оптимальный алгоритм всегда находит глобальный минимум/максимум.

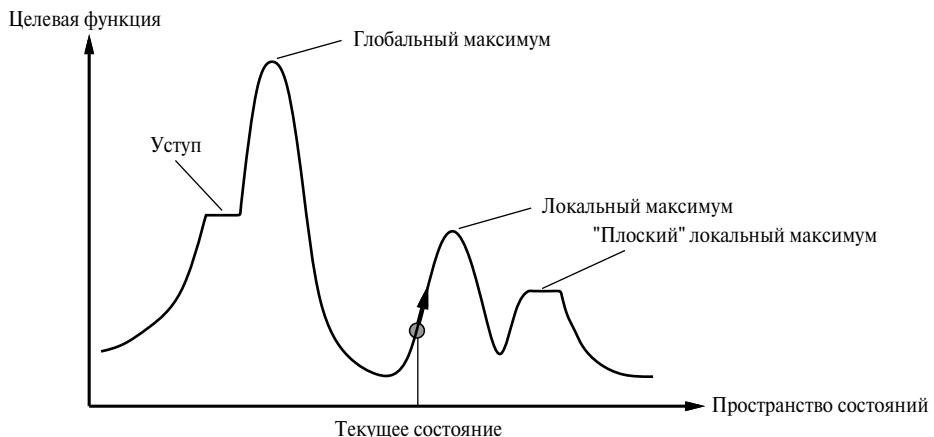


Рис. 4.7. Ландшафт одномерного пространства состояний, в котором возвышение соответствует целевой функции. Задача состоит в поиске глобального максимума. Как обозначено стрелкой, в процессе поиска по принципу “подъема к вершине” осуществляются попытки модификации текущего состояния в целях его улучшения. Различные топографические особенности ландшафта определены в тексте

### Поиск с восхождением к вершине

Алгоритм поиска с восхождением к вершине показан в листинге 4.2. Он представляет собой обычный цикл, в котором постоянно происходит перемещение в направлении возрастания некоторого значения, т.е. подъем. Работа этого алгоритма заканчивается после достижения “пика”, в котором ни одно из соседних состояний не имеет более высокого значения. В данном алгоритме не предусмотрено сопрово-

ждение дерева поиска, поэтому в структуре данных текущего узла необходимо регистрировать только состояние и соответствующее ему значение целевой функции. В алгоритме с восхождением к вершине не осуществляется прогнозирование за пределами состояний, которые являются непосредственно соседними по отношению к текущему состоянию. Это напоминает попытку альпиниста, страдающего от амнезии, найти вершину горы Эверест в густом тумане.

**Листинг 4.2. Алгоритм поиска с восхождением к вершине (версия с наиболее крутым подъемом),** который представляет собой самый фундаментальный метод локального поиска. На каждом этапе текущий узел заменяется наилучшим соседним узлом; в данной версии таковым является узел с максимальным значением *Value*, но если используется эвристическая оценка стоимости *h*, то может быть предусмотрен поиск соседнего узла с минимальным значением *h*

---

```

function Hill-Climbing(problem) returns состояние, которое
    представляет собой локальный максимум
    inputs: problem, задача
    local variables: current, узел
                      neighbor, узел

    current  $\leftarrow$  Make-Node(Initial-State[problem])
    loop do
        neighbor  $\leftarrow$  преемник узла current с наивысшим значением
        if Value[neighbor]  $\leq$  Value[current] then return
            State[current]
        current  $\leftarrow$  neighbor

```

---

Для иллюстрации поиска с восхождением к вершине воспользуемся **задачей с восемью ферзями**, которая представлена на с. 118. В алгоритмах локального поиска обычно применяется **формулировка полного состояния**, в которой в каждом состоянии на доске имеется восемь ферзей, по одному ферзю в каждом столбце. Функция определения преемника возвращает все возможные состояния, формируемые путем перемещения отдельного ферзя в другую клетку одного и того же столбца (поэтому каждое состояние имеет  $8 \times 7 = 56$  преемников). Эвристическая функция стоимости *h* определяет количество пар ферзей, которые атакуют друг друга прямо, либо косвенно (атака называется косвенной, если на одной горизонтали, вертикали или диагонали стоят больше двух ферзей). Глобальный минимум этой функции становится равным нулю, и это происходит только в идеальных решениях. На рис. 4.8, *a* показано состояние со значением *h*=17. На этом рисунке также показаны значения всех преемников данного состояния, притом что наилучшие преемники имеют значение *h*=12. Алгоритмы с восхождением к вершине обычно предусматривают случайный выбор в множестве наилучших преемников, если количество преемников больше одного.

Поиск с восхождением к вершине иногда называют **жадным локальным поиском**, поскольку в процессе его выполнения происходит захват самого хорошего соседнего состояния без предварительных рассуждений о том, куда следует отправиться дальше. Жадность считается одним из семи смертных грехов, но, как оказалось, жадные алгоритмы часто показывают весьма высокую производительность. Во время поиска с восхождением к вершине зачастую происходит очень быстрое продвижение в направлении к решению, поскольку обычно бывает чрезвычайно легко

улучшить плохое состояние. Например, из состояния, показанного на рис. 4.8, *a*, достаточно сделать лишь пять ходов, чтобы достичь состояния, показанного на рис. 4.8, *б*, которое имеет оценку  $h=1$  и очень близко к одному из решений.

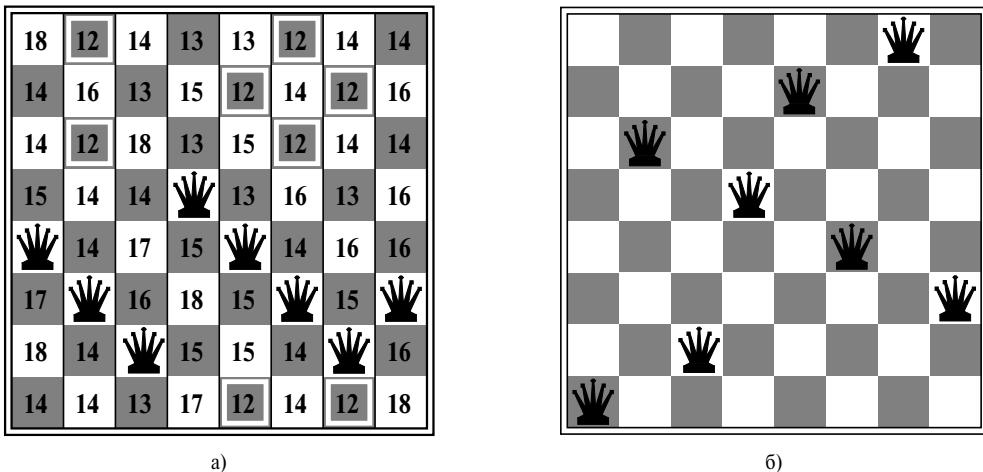
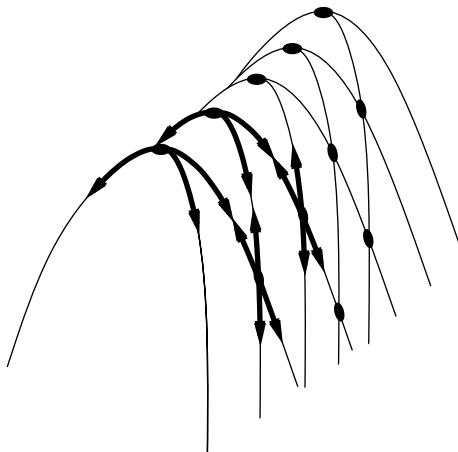


Рис. 4.8. Пример применения алгоритма с восхождением к вершине: диаграмма состояния задачи с восемью ферзями, характеризующегося эвристической оценкой стоимости  $h=17$ ; на этой диаграмме показано значение  $h$  для каждого возможного преемника, полученное путем передвижения ферзя в пределах своего столбца; отмечены наилучшие ходы (*а*); локальный минимум в пространстве состояний задачи с восемью ферзями; это состояние имеет оценку  $h=1$ , но каждый преемник характеризуется более высокой стоимостью (*б*)

К сожалению, поиск с восхождением к вершине часто заходит в тупик по описанным ниже причинам.

- **Локальные максимумы.** Локальный максимум представляет собой пик, более высокий по сравнению с каждым из его соседних состояний, но более низкий, чем глобальный максимум. Алгоритмы с восхождением к вершине, которые достигают окрестностей локального максимума, обеспечивают продвижение вверх, к этому пику, но после этого заходят в тупик, из которого больше некуда двигаться. Такая проблема схематически показана на рис. 4.7. Более конкретный пример состоит в том, что состояние, показанное на рис. 4.8, *б*, фактически представляет собой локальный максимум (т.е. локальный минимум для оценки стоимости  $h$ ); задача еще не решена, а при любом передвижении отдельно взятого ферзя ситуация становится еще хуже).
- **Хребты.** Пример хребта показан на рис. 4.9. При наличии хребтов возникают последовательности локальных максимумов, задача прохождения которых для жадных алгоритмов является очень трудной.
- **Плато.** Это область в ландшафте пространства состояний, в которой функция оценки является плоской. Оно может представлять собой плоский локальный максимум, из которого не существует выхода вверх, или ~~уступ~~ уступ, из которого возможно дальнейшее успешное продвижение (см. рис. 4.7). Поиск с восхождением к вершине может оказаться неспособным выйти за пределы плато.



*Рис. 4.9. Иллюстрация того, почему хребты вызывают сложности при восхождении к вершине. На хребет, возвышающийся слева направо, налагается решетка состояний (полусирные дуги), создавая последовательность локальных максимумов, которые не являются непосредственно связанными друг с другом. В каждом локальном максимуме все доступные действия направлены вниз*

В каждом из этих случаев рассматриваемый алгоритм достигает такой точки, из которой не может осуществляться дальнейшее успешное продвижение. Начиная со случайно сформированного состояния с восемью ферзями, алгоритм поиска с восхождением к вершине по самому кругому подъему заходит в тупик в 86% случаях, решая только 14% экземпляров этой задачи. Но он работает очень быстро, выполняя в среднем только 4 этапа в случае успешного завершения и 3 этапа, когда заходит в тупик. Это не очень плохой результат для пространства состояний с  $8^8 \approx 17$  миллионами состояний.

Алгоритм, приведенный в листинге 4.2, останавливается, достигнув плато, на котором наилучший преемник имеет такое же значение, как и в текущем состоянии. Имеет ли смысл продолжать движение, разрешив **движение в сторону** в надежде на то, что это плато в действительности представляет собой уступ, как показано на рис. 4.7? Ответ на этот вопрос обычно является положительным, но необходимо соблюдать осторожность. Если будет всегда разрешено движение в сторону, притом что движение вверх невозможно, могут возникать бесконечные циклы после того, как алгоритм достигнет плоского локального максимума, не являющегося уступом. Одно из широко применяемых решений состоит в том, что устанавливается предел количества допустимых последовательных движений в сторону. Например, можно разрешить, допустим, 100 последовательных движений в сторону в задаче с восемью ферзями. В результате этого относительное количество экземпляров задачи, решаемых с помощью восхождения к вершине, возрастает с 14 до 94%. Но за этот успех приходится платить: алгоритм в среднем выполняет приблизительно 21 этап при каждом успешном решении экземпляра задачи и 64 этапа при каждой неудаче.

Разработано много вариантов поиска с восхождением к вершине. При **стochastic search with ascent to the peak** осуществляется выбор слу-

чайным образом одного из движений вверх; вероятность такого выбора может зависеть от крутизны движения вверх. Обычно этот алгоритм сходится более медленно по сравнению с вариантом, предусматривающим наиболее крутой подъем, но в некоторых ландшафтах состояний он находит лучшие решения. При **поиске с восхождением к вершине с выбором первого варианта** реализуется стохастический поиск с восхождением к вершине путем формирования преемников случайным образом до тех пор, пока не будет сформирован преемник, лучший по сравнению с текущим состоянием. Это — хорошая стратегия, если любое состояние имеет большое количество преемников (измеряемое тысячами). В упр. 4.16 предлагается исследовать этот алгоритм.

Алгоритмы с восхождением к вершине, описанные выше, являются неполными, поскольку часто оказываются неспособными найти цель, притом что она существует, из-за того, что могут зайти в тупик, достигнув локального максимума. **Поиск с восхождением к вершине и перезапуском случайным образом** руководствуется широко известной рекомендацией: “Если первая попытка оказалась неудачной, пробуйте снова и снова”. В этом алгоритме предусмотрено проведение ряда поисков из сформированных случайным образом начальных состояний<sup>8</sup> и остановов после достижения цели. Он является полным с вероятностью, достигающей 1, даже по той тривиальной причине, что в нем в конечном итоге в качестве начального состояния формируется одно из целевых состояний. Если вероятность успеха каждого поиска с восхождением к вершине равна  $p$ , то ожидаемое количество требуемых перезапусков составляет  $1/p$ . Для экземпляров задачи с восемью ферзями, если не разрешено движение в сторону,  $p \approx 0.14$ , поэтому для нахождения цели требуется приблизительно 7 итераций (6 неудачных и 1 успешная). Ожидаемое количество этапов решения равно стоимости одной успешной итерации, которая складывается с увеличенной в  $(1-p)/p$  раз стоимостью неудачи, или составляет приблизительно 22 этапа. Если разрешено движение в сторону, то в среднем требуется  $1/0.94 \approx 1.06$  итераций и  $(1 \times 21) + (0.06/0.94) \times 64 \approx 25$  этапов. Поэтому алгоритм поиска с восхождением к вершине и перезапуском случайным образом действительно является очень эффективным применительно к задаче с восемью ферзями. Даже для варианта с тремя миллионами ферзей этот подход позволяет находить решения меньше чем за минуту<sup>9</sup>.

Успех поиска с восхождением к вершине в значительной степени зависит от формы ландшафта пространства состояний: если в нем есть лишь немного локальных максимумов и плато, то поиск с восхождением к вершине и перезапуском случайным образом позволяет очень быстро найти хорошее решение. С другой стороны, многие реальные задачи имеют ландшафт, который больше напоминает семейство дикобразов на плоском полу, где на вершине иглы каждого дикобраза живут другие миниатюрные дикобразы и т.д., до бесконечности. NP-трудные задачи обычно имеют экспоненциальное количество локальных максимумов, способных завести

<sup>8</sup> Может оказаться трудной даже сама задача формирования случайным образом некоторого состояния из неявно заданного пространства состояний.

<sup>9</sup> В [958] доказано, что в некоторых случаях лучше всего осуществлять перезапуск рандомизированного алгоритма поиска по истечении конкретной, постоянной продолжительности времени и что такой подход может оказаться гораздо более эффективным по сравнению с тем, когда разрешено продолжать каждый поиск до бесконечности. Примером такого подхода является также запрещение или ограничение количества движений в сторону.

алгоритм в тупик. Несмотря на это, часто существует возможность найти достаточно хороший локальный максимум после небольшого количества перезапусков.

### Поиск с эмуляцией отжига

Для любого алгоритма с восхождением к вершине, который никогда не выполняет движения “вниз по склону”, к состояниям с более низкой оценкой (или более высокой стоимостью), гарантируется, что он окажется неполным, поскольку такой алгоритм всегда способен зайти в тупик, достигнув локального максимума. В отличие от этого алгоритм с чисто случайным блужданием (т.е. с перемещением к преемнику, выбираемому на равных правах случайным образом из множества преемников) является полным, но чрезвычайно неэффективным. Поэтому представляется разумной попытка скомбинировать каким-то образом восхождение к вершине со случайным блужданием, что позволит обеспечить и эффективность, и полноту. Алгоритмом такого типа является алгоритм с **эмуляцией отжига**. В металлургии **отжигом** называется процесс, применяемый для отпуска металла и стекла путем нагревания этих материалов до высокой температуры, а затем постепенного охлаждения, что позволяет перевести обрабатываемый материал в низкоэнергетическое кристаллическое состояние. Чтобы понять суть эмуляции отжига, переведем наше внимание с восхождения к вершине на **градиентный спуск** (т.е. минимизацию стоимости) и представим себе, что наше задание — загнать теннисный шарик в самую глубокую лунку на неровной поверхности. Если бы мы просто позволили шарику катиться по этой поверхности, то он застрял бы в одном из локальных минимумов. А встряхивая поверхность, можно вытолкнуть шарик из локального минимума. Весь секрет состоит в том, что поверхность нужно трясти достаточно сильно, чтобы шарик можно было вытолкнуть из локальных минимумов, но не настолько сильно, чтобы он вылетел из глобального минимума. Процесс поиска решения с эмуляцией отжига заключается в том, что вначале происходит интенсивное встряхивание (аналогичное нагреву до высокой температуры), после чего интенсивность встряхивания постепенно уменьшается (что можно сравнить с понижением температуры).

Самый внутренний цикл алгоритма с эмуляцией отжига (листинг 4.3) полностью аналогичен циклу алгоритма с восхождением к вершине, но в нем вместо наилучшего хода выполняется случайно выбранный ход. Если этот ход улучшает ситуацию, то всегда принимается. В противном случае алгоритм принимает данный ход с некоторой вероятностью, меньшей 1. Эта вероятность уменьшается экспоненциально с “ухудшением” хода — в зависимости от величины  $\Delta E$ , на которую ухудшается его оценка. Кроме того, вероятность уменьшается по мере снижения “температуры”  $T$ : “плохие” ходы скорее всего могут быть разрешены в начале, когда температура высока, но становятся менее вероятными по мере снижения  $T$ . Можно доказать, что если в графике `schedule` предусмотрено достаточно медленное снижение  $T$ , то данный алгоритм позволяет найти глобальный оптимум с вероятностью, приближающейся к 1.

На первых порах, в начале 1980-х годов, поиск с эмуляцией отжига широко использовался для решения задач компоновки СБИС. Кроме того, этот алгоритм нашел широкое применение при решении задач планирования производства и других крупномасштабных задач оптимизации. В упр. 4.16 предлагается сравнить его производительность с производительностью поиска с восхождением к вершине и перезапуском случайнным образом при решении задачи с  $n$  ферзями.

**Листинг 4.3.** Алгоритм поиска с эмуляцией отжига, который представляет собой одну из версий алгоритма стохастического поиска с восхождением к вершине, в которой разрешены некоторые ходы вниз. Ходы вниз принимаются к исполнению с большей вероятностью на ранних этапах выполнения графика отжига, а затем, по мере того как проходит время, выполняются менее часто. Входной параметр `schedule` определяет значение температуры  $T$  как функции от времени

---

```

function Simulated-Annealing(problem, schedule) returns состояние
    решения
    inputs: problem, задача
              schedule, отображение между временем и "температурой"
    local variables: current, узел
                      next, узел
                      T, "температура", от которой зависит вероятность
                      шагов вниз

    current  $\leftarrow$  Make-Node(Initial-State[problem])
    for t  $\leftarrow$  1 to  $\infty$  do
        T  $\leftarrow$  schedule[t]
        if T = 0 then return current
        next  $\leftarrow$  случайно выбранный преемник состояния current
         $\Delta E \leftarrow$  Value[next] - Value[current]
        if  $\Delta E > 0$  then current  $\leftarrow$  next
        else current  $\leftarrow$  next с вероятностью только  $e^{\Delta E/T}$ 
```

---

## Локальный лучевой поиск

Стремление преодолеть ограничения, связанные с нехваткой памяти, привело к тому, что в свое время предпочтение отдавалось алгоритмам, предусматривающим хранение в памяти только одного узла, но, как оказалось, такой подход часто является слишком радикальным способом экономии памяти. В алгоритме **локального лучевого поиска**<sup>10</sup> предусмотрено отслеживание  $k$  состояний, а не только одного состояния. Работа этого алгоритма начинается с формирования случайным образом  $k$  состояний. На каждом этапе формируются все преемники всех  $k$  состояний. Если какой-либо из этих преемников соответствует целевому состоянию, алгоритм останавливается. В противном случае алгоритм выбирает из общего списка  $k$  наилучших преемников и повторяет цикл.

На первый взгляд может показаться, что локальный лучевой поиск с  $k$  состояниями представляет собой не что иное, как выполнение  $k$  перезапусков случайнym образом, но не последовательно, а параллельно. Тем не менее в действительности эти два алгоритма являются полностью разными. При поиске с перезапуском случайным образом каждый процесс поиска осуществляется независимо от других. *А в локальном лучевом поиске полезная информация передается по  $k$  параллельным потокам поиска.* Например, если в одном состоянии вырабатывается несколько хороших преемников, а во всех других  $k-1$  состояниях вырабатываются плохие преемники, то возникает такой эффект, как если бы поток, контролирующий первое состояние, сообщил другим потокам: "Идите все сюда, здесь

---

<sup>10</sup> Локальный лучевой поиск является адаптацией **лучевого поиска**, который представляет собой алгоритм, основанный на использовании пути.

трава зеленее!” Этот алгоритм способен быстро отказаться от бесплодных поисков и перебросить свои ресурсы туда, где достигнут наибольший прогресс.

В своей простейшей форме локальный лучевой поиск может страдать от отсутствия разнообразия между  $k$  состояниями, поскольку все эти состояния способны быстро сосредоточиться в небольшом регионе пространства состояний, в результате чего этот поиск начинает ненамного отличаться от дорогостоящей версии поиска с восхождением к вершине. Этот недостаток позволяет устраниТЬ вариант, называемый **стochasticеским лучевым поиском**, который аналогичен стохастическому поиску с восхождением к вершине. При стохастическом лучевом поиске вместо выбора наилучших  $k$  преемников из пула преемников-кандидатов происходит выбор  $k$  преемников случайным образом, притом что вероятность выбора данного конкретного преемника представляет собой возрастающую функцию значения его оценки. Стохастический лучевой поиск имеет некоторое сходство с процессом естественного отбора, в котором “преемники” (потомки) некоторого “состояния” (организма) образуют следующее поколение в соответствии со “значением их оценки” (в соответствии с их жизненной пригодностью).

### Генетические алгоритмы

**Генетический алгоритм** (Genetic Algorithm — GA) представляет собой вариант стохастического лучевого поиска, в котором состояния-преемники формируются путем комбинирования двух родительских состояний, а не посредством модификации единственного состояния. В нем просматривается такая же аналогия с естественным отбором, как и в стохастическом лучевом поиске, за исключением того, что теперь мы имеем дело с половым, а не бесполым воспроизведением.

Как и при лучевом поиске, работа алгоритмов GA начинается с множества  $k$  сформированных случайным образом состояний, называемых **популяцией**. Каждое состояние, или **индивидуум**, представлено в виде строки символов из конечного алфавита, чаще всего в виде строки из нулей (0) и единиц (1). Например, состояние задачи с восемью ферзями должно определять позиции восьми ферзей, каждый из которых стоит на вертикали с 8 клетками, и поэтому для его представления требуется  $8 \times \log_2 8 = 24$  бита. Иным образом, каждое состояние может быть представлено в виде восьми цифр, каждая из которых находится в диапазоне от 1 до 8. (Как будет показано ниже, эти две кодировки проявляют себя в ходе поиска по-разному.) На рис. 4.10, *a* показана популяция из четырех восьмисимвольных строк, представляющих состояния с восемью ферзями.

Процесс выработки следующего поколения состояний показан на рис. 4.10, *b–d*. На рис. 4.10, *b* каждое состояние классифицируется с помощью функции оценки, или (в терминологии GA) **функции пригодности**. Функция пригодности должна возвращать более высокие значения для лучших состояний, поэтому в задаче с восемью ферзями используется количество неатакующих друг друга пар ферзей, которое в любом решении имеет значение 28. Для этих четырех состояний соответствующие значения равны 24, 23, 20 и 11. В данном конкретном варианте генетического алгоритма вероятность выбора для воспроизведения прямо пропорциональна оценке пригодности; соответствующие вероятности в процентах показаны рядом с исходными оценками.

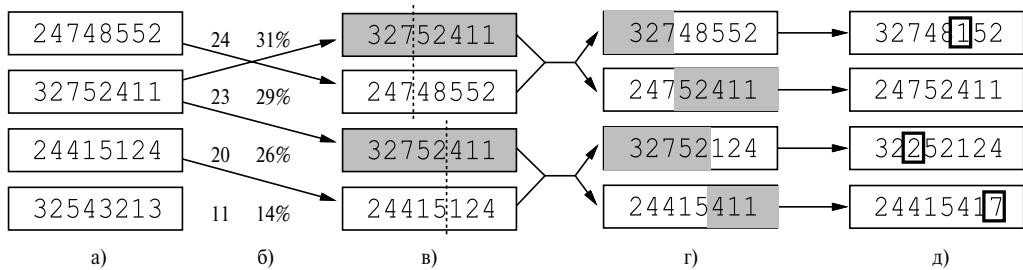


Рис. 4.10. Генетический алгоритм: начальная популяция (а); функция пригодности (б); отбор (в); скрещивание (г); мутация (д). Начальная популяция классифицируется с помощью функции пригодности, в результате чего формируются пары для скрещивания. Эти пары производят потомков, которые в конечном итоге подвергаются мутации

Как показано на рис. 4.10, *в*, для воспроизведения случайным образом выбираются две пары в соответствии с вероятностями, показанными на рис. 4.10, *б*. Обратите внимание на то, что один индивидуум выбирается дважды, а один вообще остается не выбранным<sup>11</sup>. Для каждой пары, предназначеннной для воспроизведения, среди позиций в строке случайным образом выбирается точка скрещивания. На рис. 4.10 точки скрещивания находятся после третьей цифры в первой паре и после пятой цифры во второй паре<sup>12</sup>.

Как показано на рис. 4.10, *г*, сами потомки создаются путем перекрестного обмена подстроками родительских строк, разорванных в точке скрещивания. Например, первый потомок первой пары получает три первые цифры от первого родителя, а остальные цифры — от второго родителя, тогда как второй потомок получает первые три цифры от второго родителя, а остальные — от первого родителя. Состояния задачи с восемью ферзями, участвующие в этом этапе воспроизведения, показаны на рис. 4.11. Данный пример иллюстрирует тот факт, что если два родительских состояния являются весьма различными, то операция скрещивания способна выработать состояние, которое намного отличается и от одного, и от другого родительского состояния. Часто происходит так, что популяция становится весьма разнообразной на самых ранних этапах этого процесса, поэтому скрещивание (как и эмуляция отжига) в основном обеспечивает выполнение крупных этапов в пространстве состояний в начале процесса поиска и более мелких этапов позднее, когда большинство индивидуумов становится весьма похожими друг на друга.

Наконец, на рис. 4.10, *д* показано, что каждое местонахождение подвергается случайной мутации с небольшой независимой вероятностью. В первом, третьем и четвертом потомках мутация свелась к изменению одной цифры. В задаче с восемью ферзями эта операция соответствует выбору случайным образом одного ферзя и пе-

<sup>11</sup> Существует много вариантов этого правила выбора. Можно показать, что при использовании метода отсеивания, в котором отбрасываются все индивидуумы с оценками ниже заданного порога, алгоритм сходится быстрее, чем при использовании версии со случайнм выбором [81].

<sup>12</sup> Кодировка начинает играть решающую роль в процессе решения именно на этом этапе. Если вместо 8 цифр используется 24-битовая кодировка, то вероятность попадания точки скрещивания в середину цифры становится равной  $2/3$ , а это приводит к тому, что мутация этой цифры по сути становится произвольной.

ремещению этого ферзя на случайно выбранную клетку в его столбце. В листинге 4.4 приведен алгоритм, который реализует все эти этапы.

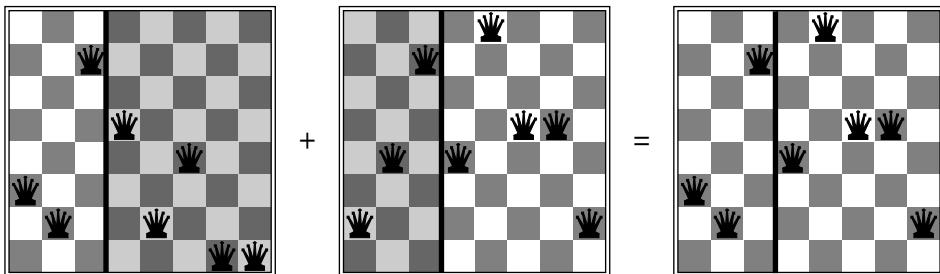


Рис. 4.11. Состояния задачи с восемью ферзями, соответствующие первым двум родительским состояниям, показанным на рис. 4.10, в, и первому потомку, показанному на рис. 4.10, г. Затененные столбцы на этапе скрещивания теряются, а незатененные сохраняются

**Листинг 4.4. Генетический алгоритм.** Этот алгоритм аналогичен тому, который показан схематически на рис. 4.10, за одним исключением: в этой более широко применяемой версии алгоритма каждое скрещивание двух родителей приводит к получению только одного потомка, а не двух

---

```

function Genetic-Algorithm(population, Fitness-Fn) returns индивидуум
    individual
    inputs: population, множество индивидуумов (популяция)
              Fitness-Fn, функция, которая измеряет пригодность
              индивидуума

    repeat
        new_population  $\leftarrow$  пустое множество
        loop for i from 1 to Size(population) do
            x  $\leftarrow$  Random-Selection(population, Fitness-Fn)
            y  $\leftarrow$  Random-Selection(population, Fitness-Fn)
            child  $\leftarrow$  Reproduce(x, y)
            if (небольшое случайно выбранное значение вероятности)
                then child  $\leftarrow$  Mutate(child)
                добавить дочерний индивидуум child
                к множеству new_population
        population  $\leftarrow$  new_population
    until некоторый индивидуум не станет достаточно пригодным
           или не истечет достаточно количество времени
    return наилучший индивидуум individual в популяции population,
           в соответствии с функцией Fitness-Fn

function Reproduce(x, y) returns индивидуум individual
    inputs: x, y, индивидуумы-родители

    n  $\leftarrow$  Length(x)
    c  $\leftarrow$  случайное число от 1 до n
    return Append(Substring(x, 1, c), Substring(y, c+1, n))

```

---

## ЭВОЛЮЦИЯ И ПОИСК

Теория **эволюции** была изложена Чарльзом Дарвином в его книге *On the Origin of Species by Means of Natural Selection* (Происхождение видов путем естественного отбора) [325]. Основная идея этой теории проста: при воспроизведстве возникают вариации (известные как **мутации**), которые сохраняются в следующих поколениях с частотой, приблизительно пропорциональной тому, как они влияют на пригодность к воспроизведству.

Дарвин разрабатывал свою теорию, не зная о том, благодаря чему происходит наследование и модификация характерных особенностей организмов. Вероятностные законы, управляющие этими процессами, были впервые обнаружены Грегором Менделем [1034], монахом, проводившим эксперименты с душистым горошком с использованием метода, названного им *искусственным оплодотворением*. Гораздо позже Уотсон и Крик [1559] выявили структуру молекулы ДНК (дезоксирибонуклеиновой кислоты) и ее алфавит, состоящий из аминокислот АГТЦ (аденин, гуанин, тимин, цитозин). В предложенной ими стандартной модели вариации возникают и в результате точечных мутаций в последовательности этих аминокислот, и в результате “скрещивания” (при котором ДНК потомка формируется путем объединения длинных секций ДНК от каждого родителя).

Аналогия между этим процессом и алгоритмами локального поиска уже была описана выше; принципиальное различие между стохастическим лучевым поиском и эволюцией состоит в использовании полового воспроизведения, в котором потомки формируются с участием нескольких организмов, а не только одного. Однако фактически механизмы эволюции намного богаче по сравнению с тем, что допускает большинство генетических алгоритмов. Например, мутации могут быть связаны с обращением, дублированием и перемещением больших фрагментов ДНК; некоторые вирусы заимствуют ДНК из одного организма и вставляют в другой; кроме того, существуют взаимозаменяемые гены, которые лишь копируют самих себя в геноме много тысяч раз. Есть даже такие гены, которые отправляют клетки потенциальных родителей, не несущие ген этого типа, повышая тем самым шансы на воспроизведение данного гена. Наиболее важным является тот факт, что гены сами кодируют те механизмы, с помощью которых геном воспроизводится и транслируется в организме. В генетических алгоритмах такие механизмы представляют собой отдельную программу, не закодированную в строках, с которыми осуществляются манипуляции.

На первый взгляд может показаться, что механизм дарвиновской эволюции является неэффективным, поскольку в его рамках на Земле за многие тысячи лет было вслепую сформировано около  $10^{45}$  или примерно столько организмов, притом что за такое время поисковые эвристики этого механизма не улучшились ни на йоту. Но за пятьдесят лет до Дарвина французский натуралист Жан Ламарк [884], получивший известность в другой области, предложил теорию эволюции, согласно которой характерные особенности организма, приобретенные в результате его адаптации на протяжении срока жизни этого организма, передаются его потомкам. Такой процесс был бы очень эффективным, но в природе, по-видимому, не происходит. Намного

позднее Джеймс Болдуин [64] предложил теорию, которая внешне кажется аналогичной; согласно этой теории, поведение, которому обучился организм на протяжении своей жизни, способствует повышению скорости эволюции. В отличие от теории Ламарка, теория Болдуина полностью согласуется с дарвиновской теорией эволюции, поскольку в ней учитываются давления отбора, действующие на индивидуумов, которые нашли локальные оптимумы среди множества возможных вариантов поведения, допустимых согласно их генетической природе. Современные компьютерные модели подтвердили, что “эффект Болдуина” является реальным, при условии, что “обычная” эволюция способна создавать организмы, внутренние показатели производительности которых каким-то образом коррелируют с их фактической жизненной пригодностью.

Как и в алгоритмах стохастического лучевого поиска, в генетических алгоритмах тенденция к стремлению к максимуму сочетается с проводимым случайным образом исследованием и с обменом информацией между параллельными поисковыми потоками. Основное преимущество генетических алгоритмов (если таковое действительно имеется) связано с применением операции скрещивания. Тем не менее можно доказать математически, что если позиции в генетическом коде первоначально устанавливаются в случайном порядке, то скрещивание не дает никаких преимуществ. На интуитивном уровне можно предположить, что преимуществом была бы способность комбинировать в результате скрещивания большие блоки символов, которые были независимо сформированы в ходе эволюции для выполнения полезных функций, что позволило бы повысить степень детализации, с которой действует этот алгоритм поиска. Например, преимущество достигалось бы, если в качестве полезного блока была выделена строка символов, предусматривающая размещение первых трех ферзей в позициях 2, 4 и 6 (где они не атакуют друг друга), после чего можно было бы объединять этот блок с другими блоками для формирования решения.

В теории генетических алгоритмов описано, как действует этот подход, в котором используется идея  $\bowtie$  схемы, представляющей собой такую подстроку, где некоторые из позиций могут оставаться не заданными. Например, схема 246\* \* \* \* описывает такие состояния всех восьми ферзей, в которых первые три ферзы находятся соответственно в позициях 2, 4 и 6. Строки, соответствующие этой схеме (такие как 24613578), называются экземплярами схемы. Можно доказать, что если средняя пригодность экземпляров некоторой схемы находится выше среднего, то количество экземпляров этой схемы в популяции будет расти со временем. Очевидно, что данный эффект вряд ли окажется существенным, если смежные биты полностью не связаны друг с другом, поскольку в таком случае будет лишь немного непрерывных блоков, которые предоставляли бы какое-то постоянное преимущество. Генетические алгоритмы работают лучше всего в сочетании со схемами, соответствующими осмысленным компонентам решения. Например, если строка представляет какую-либо радиотехническую антенну, то схемы могут соответствовать компонентам этой антенны, таким как рефлекторы и дефлекторы. Хороший компонент, по всей вероятности, будет оставаться хорошим во многих разных проектах. Из этого следует, что для успешного использования генетических алгоритмов требуется тщательное конструирование представления задачи.

На практике генетические алгоритмы оказали глубокое влияние на научные методы, применяющиеся при решении таких задач оптимизации, как компоновка электронных схем и планирование производства. В настоящее время уже не так ясно, вызвана ли притягательность генетических алгоритмов их высокой производительностью или обусловлена эстетически привлекательными истоками в теории эволюции. Но все еще необходимо проделать большой объем работы для выяснения условий, при которых генетические алгоритмы обладают наиболее высокой производительностью.

#### 4.4. ЛОКАЛЬНЫЙ ПОИСК В НЕПРЕРЫВНЫХ ПРОСТРАНСТВАХ

В главе 2 было описано различие между дискретными и непрерывными вариантами среды, а также указано, что большинство реальных вариантов среды являются непрерывными. Но еще ни один из описанных выше алгоритмов не способен действовать в непрерывных пространствах состояний, поскольку в этих алгоритмах в большинстве случаев функция определения преемника возвращала бы бесконечно большое количество состояний! В настоящем разделе приведено очень краткое введение в некоторые методы локального поиска, предназначенные для нахождения оптимальных решений в непрерывных пространствах. Литература по этой теме весьма обширна; многие из этих основных методов впервые были созданы в XVII веке после разработки первых математических исчислений Ньютоном и Лейбницием<sup>13</sup>. Применение данных методов рассматривается в нескольких главах настоящей книги, включая главы, касающиеся обучения, машинного зрения и робототехники. Короче говоря, эти методы касаются всего, что связано с реальным миром.

Начнем изложение этой темы с примера. Предположим, что где-то в Румынии требуется найти место для размещения трех новых аэропортов таким образом, чтобы сумма квадратов расстояний от каждого города на карте (см. рис. 3.1) до ближайшего к нему аэропорта была минимальной. В таком случае пространство состояний определено координатами аэропортов:  $(x_1, y_1)$ ,  $(x_2, y_2)$  и  $(x_3, y_3)$ . Это — шестимерное пространство; иными словами можно выразить данную мысль так, что состояния определяются шестью **переменными**. (Вообще говоря, состояния определяются **n**-мерным вектором переменных,  $\mathbf{x}$ .) Перемещение в этом пространстве соответствует переносу одного или нескольких из этих аэропортов в другое место на карте. Целевую функцию  $f(x_1, y_1, x_2, y_2, x_3, y_3)$  после определения ближайших городов можно вычислить довольно легко, но гораздо сложнее составить общее выражение, соответствующее искомому решению.

Один из способов предотвращения необходимости заниматься непрерывными задачами состоит в том, чтобы просто **дискретизировать** окрестности каждого состояния. Например, можно предусмотреть перемещение одновременно только одного аэропорта в направлении либо  $x$ , либо  $y$  на постоянную величину  $\pm\Delta$ . При наличии шести переменных это соответствует двенадцати возможным преемникам для каждого состояния. После этого появляется возможность применить любой из алгоритмов локального поиска, описанных выше. Кроме того, алгоритмы стохастиче-

<sup>13</sup> Для чтения этого раздела желательно обладать элементарными знаниями о системах исчисления в многомерном пространстве и векторной арифметике.

ского поиска с восхождением к вершине и поиска с эмуляцией отжига можно применять непосредственно, без дискретизации этого пространства. Такие алгоритмы выбирают преемников случайным образом, что может быть осуществлено путем формирования случайным образом векторов с длиной  $\Delta$ .

Имеется много методов, в которых при осуществлении попыток найти максимум используется **градиент** ландшафта. Градиент целевой функции представляет собой вектор  $\nabla f$ , позволяющий определить величину и направление наиболее крутого склона. Для рассматриваемой задачи справедливо следующее соотношение:

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

В некоторых случаях можно найти максимум, решая уравнение  $\nabla f=0$ . (Это можно сделать, например, при размещении только одного аэропорта; решение представляет собой арифметическое среднее координат всех городов.) Но во многих случаях это уравнение не может быть решено в замкнутой форме. Например, при наличии трех аэропортов выражение для градиента зависит от того, какие города являются ближайшими к каждому аэропорту в текущем состоянии. Это означает, что мы можем вычислить этот градиент локально, но не глобально. Даже в таком случае остается возможность выполнять поиск с восхождением к вершине по самому крутому склону, обновляя текущее состояние с помощью следующей формулы:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

где  $\alpha$  — небольшая константа. В других случаях целевая функция в дифференцируемой форме может оказаться вообще не доступной, например, значение конкретного множества местонахождений аэропортов может быть определено в результате вызова на выполнение какого-то пакета крупномасштабного экономического моделирования. В таких случаях можно определить так называемый **эмпирический градиент**, оценивая отклик на небольшие увеличения и уменьшения каждой координаты. Поиск по эмпирическому градиенту является аналогичным поиску с восхождением к вершине по самому крутому склону в дискретизированной версии данного пространства состояний.

За фразой “ $\alpha$  — небольшая константа” скрывается огромное разнообразие методов определения значения  $\alpha$ . Основная проблема состоит в том, что если значение  $\alpha$  слишком мало, то требуется слишком много этапов поиска, а если слишком велико, то в поиске можно проскочить максимум. Попытка преодолеть эту дилемму предпринимается в методе **линейного поиска**, который предусматривает продолжение поиска в направлении текущего градиента (обычно путем повторного удвоения  $\alpha$ ) до тех пор, пока  $f$  не начнет снова уменьшаться. Точка, в которой это происходит, становится новым текущим состоянием. Сформировалось несколько научных школ, в которых доминируют разные взгляды на то, каким образом в этой точке следует выбирать новое направление.

Для многих задач наиболее эффективным алгоритмом является почтенный метод **Ньютона-Рафсона** [1132], [1266]. Это — общий метод поиска корней функций, т.е. метод решения уравнений в форме  $g(x)=0$ . Этот алгоритм действует на основе вычисления новой оценки для корня  $\mathbf{x}$  в соответствии с формулой Ньютона:

$$\mathbf{x} \leftarrow \mathbf{x} - g(\mathbf{x}) / g'(\mathbf{x})$$

Чтобы найти максимум или минимум  $f$ , необходимо найти такое значение  $\mathbf{x}$ , для которого градиент равен нулю (т.е.  $\nabla f(\mathbf{x}) = \mathbf{0}$ ). Поэтому функция  $g(\mathbf{x})$  в формуле Ньютона принимает вид  $\nabla f(\mathbf{x})$ , и уравнение обновления состояния может быть записано в матрично-векторной форме следующим образом:

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$

где  $\mathbf{H}_f(\mathbf{x})$  — представляет собой **гессиан** (матрицу вторых производных), элементы которого  $H_{ij}$  задаются выражением  $\partial^2 f / \partial x_i \partial x_j$ . Поскольку гессиан имеет  $n^2$  элементов, то метод Ньютона–Рафсона в многомерных пространствах становится дорогостоящим, поэтому было разработано множество способов приближенного решения этой задачи.

Локальные максимумы, хребты и плато создают затруднения в работе методов локального поиска не только в дискретных пространствах состояний, но и в непрерывных пространствах состояний. Для преодоления этих затруднений могут применяться алгоритмы с перезапуском случайным образом и с эмуляцией отжига, которые часто оказываются достаточно полезными. Тем не менее многомерные непрерывные пространства характеризуются очень большим объемом, в котором легко потеряться.

Последней темой, с которой необходимо кратко ознакомиться, является **оптимизация с ограничениями**. Задача оптимизации называется задачей оптимизации с ограничениями, если решения в ней должны удовлетворять некоторым жестким ограничениям на значения каждой переменной. Например, в рассматриваемой задаче с размещением аэропортов можно ввести ограничения, чтобы места для аэропортов находились в пределах Румынии, причем на суше (а не, допустим, на середине озера). Сложность задач оптимизации с ограничениями зависит от характера ограничений и от целевой функции. Наиболее широко известной категорией таких задач являются задачи **линейного программирования**, в которых ограничения должны представлять собой линейные неравенства, образующие выпуклую область, а целевая функция также является линейной. Задачи линейного программирования могут быть решены за время, полиномиально зависящее от количества переменных. Кроме того, проводились исследования задач с другими типами ограничений и целевых функций: квадратичное программирование, коническое программирование второго порядка и т.д.

## 4.5. ПОИСКОВЫЕ АГЕНТЫ, ДЕЙСТВУЮЩИЕ В ОПЕРАТИВНОМ РЕЖИМЕ, И НЕИЗВЕСТНЫЕ ВАРИАНТЫ СРЕДЫ

До сих пор в данной книге изложение было сосредоточено на описании агентов, в которых используются алгоритмы **поиска в автономном режиме**. Эти агенты вычисляют полное решение, прежде чем ступить в реальный мир (см. листинг 3.1), а затем выполняют это решение, не обращаясь к данным своих восприятий. В отличие от этого любой агент, выполняющий **поиск в оперативном режиме**<sup>14</sup>,

<sup>14</sup> В компьютерных науках термин “работающий в оперативном режиме” обычно используется по отношению к алгоритмам, которые должны обрабатывать входные данные по мере их получения, а не ожидать, пока станет доступным все множество входных данных.

функционирует по методу **чередования** вычислений и действий: вначале предпринимает действие, затем обозревает среду и вычисляет следующее действие. Поиск в оперативном режиме целесообразно применять в динамических или полудинамических проблемных областях; таковыми являются проблемные области, в которых назначается штраф за то, что агент ведет себя пассивно и вычисляет свои действия слишком долго. Еще более оправданным является использование поиска в оперативном режиме в стохастических проблемных областях. Вообще говоря, результаты любого поиска в автономном режиме должны сопровождаться экспоненциально большим планом действий в непредвиденных ситуациях, в котором учитываются все возможные варианты развития событий, тогда как при поиске в оперативном режиме необходимо учитывать лишь то, что действительно происходит. Например, агент, играющий в шахматы, должен быть настолько хорошо проконсультирован, чтобы мог сделать свой первый ход задолго до того, как станет ясен весь ход игры.

Применение поиска в оперативном режиме является необходимым при решении любой **задачи исследования**, в которой агенту не известны состояния и действия. Агент, находящийся в таком положении полного неведения, должен использовать свои действия в качестве экспериментов для определения того, что делать дальше, и поэтому вынужден чередовать вычисления и действия.

Каноническим примером применения поиска в оперативном режиме может служить робот, который помещен в новое здание и должен его исследовать, чтобы составить карту, которую может затем использовать для перехода из точки A в точку B. Примерами алгоритмов поиска в оперативном режиме являются также методы выхода из лабиринтов (как известно, такие знания всегда были нужны вдохновляющим нас на подвиги героям древности). Однако исследование пространства — это не единственная форма познания окружающего мира. Рассмотрим поведение новорожденного ребенка: в его распоряжении есть много возможных действий, но он не знает, к чему приведет выполнение какого-либо из них, а эксперименты проводит лишь в немногих возможных состояниях, которых он может достичь. Постепенное изучение ребенком того, как устроен мир, отчасти представляет собой процесс поиска в оперативном режиме.

### Задачи поиска в оперативном режиме

Любая задача поиска в оперативном режиме может быть решена только агентом, выполняющим и вычисления, и действия, а не осуществляющим лишь вычислительные процессы. Предполагается, что агент обладает только описанными ниже знаниями.

- Функция  $Actions(s)$ , которая возвращает список действий, допустимых в состоянии  $s$ .
- Функция стоимости этапа  $c(s, a, s')$ ; следует отметить, что она не может использоваться до тех пор, пока агент не знает, что результатом является состояние  $s'$ .
- Функция  $Goal-Test(s)$ .

Следует, в частности, отметить, что агент не может получить доступ к преемникам какого-либо состояния, иначе чем путем фактического опробования всех действий в этом состоянии. Например, в задаче с лабиринтом, показанной на рис. 4.12,

агент не знает, что переход в направлении *Up* из пункта  $(1, 1)$  приводит в пункт  $(1, 2)$ , а выполнив это действие, не знает, позволит ему действие *Down* вернуться назад в пункт  $(1, 1)$ . Такая степень неведения в некоторых приложениях может быть уменьшена, например, робот-исследователь может знать, как работают его действия по передвижению, и оставаться в неведении лишь в отношении местонахождения препятствий.

Мы будем предполагать, что агент всегда может распознать то состояние, которое он уже посещал перед этим, кроме того, будем руководствоваться допущением, что все действия являются детерминированными. (Последние два допущения будут ослаблены в главе 17.) Наконец, агент может иметь доступ к некоторой допустимой эвристической функции  $h(s)$ , которая оценивает расстояние от текущего состояния до целевого. Например, как показано на рис. 4.12, агент может знать местонахождение цели и быть способным использовать эвристику с манхэттенским расстоянием.

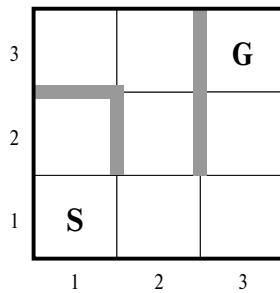


Рис. 4.12. Простая задача с лабиринтом. Агент начинает движение с квадрата  $S$  и должен достичь квадрата  $G$ , но ничего не знает о самой среде

Как правило, назначение агента состоит в том, чтобы достичь целевого состояния, минимизируя при этом стоимость. (Еще одно возможное назначение агента может заключаться в том, чтобы он просто исследовал всю эту среду.) Стоимость представляет собой суммарную стоимость пути, фактически пройденного агентом. Обычно принято сравнивать эту стоимость со стоимостью пути, по которому следовал бы агент, если бы он заранее знал пространство поиска, т.е. со стоимостью фактического кратчайшего пути (или кратчайшего полного обследования). В терминологии алгоритмов поиска в оперативном режиме это соотношение называется **коэффициентом конкурентоспособности**; желательно, чтобы этот коэффициент был как можно меньше.

Хотя такое требование по минимизации коэффициента конкурентоспособности может показаться резонным, можно легко доказать, что в некоторых случаях наилучший достижимый коэффициент конкурентоспособности (competitive ratio) является бесконечным. Например, если некоторые действия необратимы, то поиск в оперативном режиме может в конечном итоге перейти в тупиковое состояние, из которого не достижимо целевое состояние. Возможно, читатель сочтет выражение “в конечном итоге” неубедительным; в конце концов, должен же существовать такой алгоритм, который окажется способным не упираться в тупик в ходе проведения исследований с его помощью! Поэтому уточним приведенное выше утверждение таким образом: **ни один алгоритм не позволяет избежать тупиков во всех возможных**

*пространствах состояний.* Рассмотрим два пространства состояний с тупиками, показанные на рис. 4.13, а). Для алгоритма поиска в оперативном режиме, который посетил состояния  $S$  и  $A$ , оба эти пространства состояний представляются идентичными, поэтому он должен принять одинаковое решение в обоих из них. Поэтому в одном из этих состояний алгоритм потерпит неудачу. Это — один из примеров **возражения противника** (adversary argument), поскольку легко себе представить, как противник формирует пространство состояний в ходе того, как это пространство исследуется агентом, и может размещать цели и тупики везде, где пожелает.

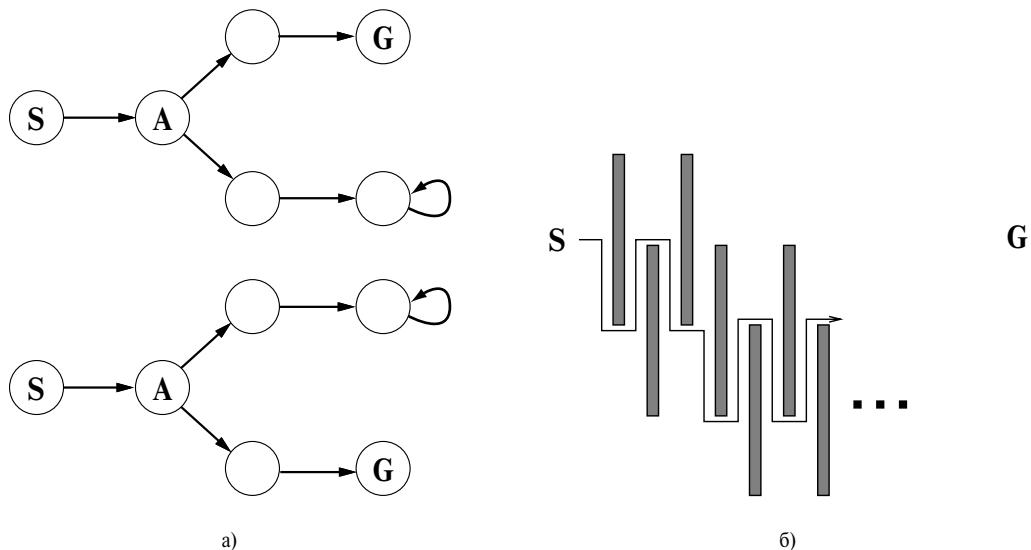


Рис. 4.13. Примеры ситуаций, характеризующихся бесконечным значением коэффициента конкурентоспособности: два пространства состояний, которые способны завести агента, выполняющего поиск в оперативном режиме, в тупик; любой конкретный агент потерпит неудачу, по меньшей мере, в одном из этих пространств (а); двухмерная среда, которая может вынудить агента, выполняющего поиск в оперативном режиме, следовать по маршруту к цели, который может оказаться сколь угодно неэффективным; какой бы выбор ни сделал агент, противник блокирует его маршрут еще одной длинной, тонкой стеной, чтобы путь, по которому следует агент, стал намного длиннее по сравнению с наилучшим возможным путем (б)

Тупики представляют собой одну из реальных сложностей в области исследований, проводимых с помощью робота, поскольку лестницы, пандусы, обрывы и многие другие формы естественного ландшафта создают предпосылки для необратимых действий. Для того чтобы добиться прогресса, мы будем предполагать, что данное пространство состояний является **безопасно исследуемым**, т.е. что некоторые целевые состояния достижимы из каждого достижимого состояния. Пространства состояний с обратимыми действиями, такие как лабиринты и задачи игры в восемь, могут рассматриваться как неориентированные графы и, вполне очевидно, являются безопасно исследуемыми.

Но даже в безопасно исследуемых вариантах среды нельзя гарантировать достижение какого-либо ограниченного значения коэффициента конкурентоспособности, если в них имеются маршруты с неограниченной стоимостью. Это утверждение легко доказать применительно к вариантам среды с необратимыми действиями, но

фактически оно остается истинным также и для обратимого случая (см. рис. 4.13, б). По этой причине обычно принято описывать производительность алгоритмов поиска в оперативном режиме с учетом размеров всего пространства состояний, а не только глубины самой поверхностной цели.

### Агенты, выполняющие поиск в оперативном режиме

После каждого действия агент, выполняющий поиск в оперативном режиме, получает результаты акта восприятия, с помощью которых может узнать, какого состояния он достиг; на основании этой информации агент может дополнить карту своей среды. Текущая карта используется для принятия решения о том, куда двигаться дальше. Такое чередование планирования и выполнения действий означает, что алгоритмы поиска в оперативном режиме весьма значительно отличаются от алгоритмов поиска в автономном режиме, которые рассматривались выше. Например, алгоритмы поиска в автономном режиме, такие как A\*, обладают способностью развертывать узел в одной части пространства, а затем немедленно развертывать узел в другой части пространства, поскольку для развертывания узла требуются моделируемые, а не реальные действия. С другой стороны, любой алгоритм поиска в оперативном режиме позволяет агенту развертывать только тот узел, который он физически занимает. Для предотвращения необходимости путешествовать через все дерево, чтобы развернуть следующий узел, представляется более удобным развертывание узлов в локальном порядке. Именно таким свойством обладает поиск в глубину, поскольку (за исключением операции возврата) следующий развертываемый узел является дочерним узлом ранее развернутого узла.

Алгоритм агента, выполняющего поиск в глубину в оперативном режиме, приведен в листинге 4.5. Агент хранит составленную им карту в таблице *result[a, s]*, в которой регистрируются состояния, явившиеся следствием выполнения действия *a* в состоянии *s*. Этот агент предпринимает попытку выполнения из текущего состояния любого действия, которое еще не было исследовано в этом состоянии. Сложности возникают после того, как агент опробовал все действия в некотором состоянии. При поиске в глубину в автономном режиме это состояние удаляется из очереди, а при поиске в оперативном режиме агент должен выполнить возврат физически. При поиске в глубину это равносильно возврату в последнее по времени состояние, из которого агент перешел в текущее состояние. Соответствующая организация работы обеспечивается за счет ведения такой таблицы, где для каждого состояния перечисляются состояния-предшественники, к которым агент еще не выполнял возврат. Если агент исчерпал список состояний, к которым может выполнить возврат, это означает, что проведенный агентом поиск является полным.

**Листинг 4.5.** Агент, выполняющий поиск в оперативном режиме, который проводит исследование с использованием поиска в глубину. Этот агент может применяться только в пространствах двунаправленного поиска

```
function Online-DFS-Agent(s') returns действие action
    inputs: s', восприятие, позволяющее идентифицировать
              текущее состояние
    static: result, таблица, индексированная по действиям и
              состояниям, первоначально пустая
```

*unexplored*, таблица, в которой для каждого посещенного состояния перечислены еще не опробованные действия  
*unbacktracked*, таблица, в которой для каждого посещенного состояния перечислены еще не опробованные возвраты  
*s, a*, предыдущие состояние и действие, первоначально неопределенные

```

if Goal-Test(s') then return stop
if s' представляет собой новое состояние
    then unexplored[s']  $\leftarrow$  Actions(s')
if s не является неопределенным then do
    result[a, s]  $\leftarrow$  s'
    добавить s в начало таблицы unbacktracked[s']
if элемент unexplored[s'] пуст then
    if элемент unbacktracked[s'] пуст then return stop
    else a  $\leftarrow$  действие b, такое что
        result[b, s'] = Pop(unbacktracked[s'])
else a  $\leftarrow$  Pop(unexplored[s'])
s  $\leftarrow$  s'
return a
```

---

Рекомендуем читателю провести трассировку хода выполнения процедуры *Online-DFS-Agent* применительно к лабиринту, показанному на рис. 4.12. Можно довольно легко убедиться в том, что в наихудшем случае агент в конечном итоге пройдет по каждой связи в данном пространстве состояний точно два раза. При решении задачи обследования такая организация работы является оптимальной; а при решении задачи поиска целей, с другой стороны, коэффициент конкурентоспособности может оказаться сколь угодно неэффективным, если агент будет отправляться в длительные экскурсии, притом что целевое состояние находится непосредственно рядом с начальным состоянием. Такая проблема решается с использованием варианта метода итеративного углубления для работы в оперативном режиме; в среде, представляющей собой однородное дерево, коэффициент конкурентоспособности подобного агента равен небольшой константе.

В связи с тем что в алгоритме *Online-DFS-Agent* используется метод с возвратами, он может применяться только в таких пространствах состояний, где действия являются обратимыми. Существуют также немного более сложные алгоритмы, применимые в пространствах состояний общего вида, но ни один из подобных алгоритмов не характеризуется ограниченным коэффициентом конкурентоспособности.

### Локальный поиск в оперативном режиме

Как и поиск в глубину, **поиск с восхождением к вершине** обладает свойством локализации применительно к операциям развертывания в нем узлов. В действительности сам поиск с восхождением к вершине уже можно считать алгоритмом поиска в оперативном режиме, поскольку предусматривает хранение в памяти только одного текущего состояния! К сожалению, в своей простейшей форме этот алгоритм не очень полезен, так как оставляет агента в таком положении, что последний не может покинуть локальный максимум и отправиться куда-то еще. Более того, в этом алго-

ритме не может использоваться перезапуск случайным образом, поскольку агент не способен перенести самого себя в новое состояние.

Вместо перезапуска случайным образом для исследования среды может быть предусмотрено использование **случайного блуждания** (random walk). В методе случайного блуждания просто выбирается случайным образом одно из действий, доступных из текущего состояния; предпочтение отдается действиям, которые еще не были опробованы. Легко доказать, что метод случайного блуждания позволяет агенту в конечном итоге найти цель или выполнить свою задачу исследования, при условии, что пространство является конечным<sup>15</sup>. С другой стороны, этот процесс может оказаться очень продолжительным. На рис. 4.14 показана среда, в которой для поиска цели с помощью метода случайного блуждания может потребоваться количество этапов, измеряемое экспоненциальной зависимостью, поскольку на каждом этапе вероятность шага назад вдвое превышает вероятность шага вперед. Безусловно, что этот пример является надуманным, но существует много реальных пространств состояний, топология которых способствует возникновению “ловушек” такого рода при случайном блуждании.

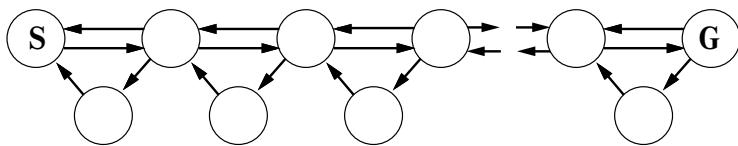


Рис. 4.14. Среда, в которой применение метода случайного блуждания для поиска цели требует выполнения такого количества этапов, которое определяется экспоненциальной зависимостью

Как оказалось, более эффективным подходом является дополнение метода поиска с восхождением к вершине способностью запоминать, а не способностью выбирать следующее действие случайным образом. Основная идея состоит в том, что необходимо хранить в памяти “текущую наилучшую оценку”  $H(s)$  стоимости достижения цели из каждого состояния, которое уже было посещено. На первых порах  $H(s)$  представляет собой только эвристическую оценку  $h(s)$  и обновляется по мере того, как агент приобретает опыт в исследовании пространства состояний. На рис. 4.15 показан простой пример одномерного пространства состояний. На рис. 4.15, а показано, что агент, по-видимому, зашел в тупик, попав в плоский локальный минимум, соответствующий затененному состоянию. Вместо того чтобы оставаться там, где находится, агент должен последовать по тому пути, который может рассматриваться как наилучший путь к цели согласно текущим оценкам стоимостей для соседних состояний. Оценка стоимости достижения цели через соседнее состояние  $s'$  равна оценке стоимости достижения  $s'$ , которая складывается с оценкой стоимости достижения цели из  $s'$ , т.е. равна  $c(s, a, s') + H(s')$ . В данном примере имеются два действия, с оценками стоимости 1+9 и 1+2, поэтому, по всей видимости, лучше всего двигаться вправо. После этого становится очевидно, что оценка стоимости для

<sup>15</sup> Этот вариант бесконечного пространства фактически является гораздо более сложным. Методы случайного блуждания являются полными в бесконечных одно- и двухмерных решетках, но не в трехмерных! В последнем случае вероятность того, что блуждающий агент в конечном итоге возвратится в начальную точку, составляет лишь около 0,3405 (для ознакомления с общим введением в эту тему см. [703]).

этого затененного состояния, равная 2, была слишком оптимистической. Поскольку стоимость наилучшего хода равна 1 и он ведет в состояние, которое находится по меньшей мере на расстоянии 2 шагов от цели, то затененное состояние должно находиться по меньшей мере на расстоянии 3 шагов от цели, поэтому его оценка  $H$  должна быть обновлена соответствующим образом, как показано на рис. 4.15, б. Продолжая этот процесс, агент передаст вперед и назад еще дважды, каждый раз обновляя оценку  $H$  и “сглаживая” локальный минимум до тех пор, пока не вырвется из него вправо.

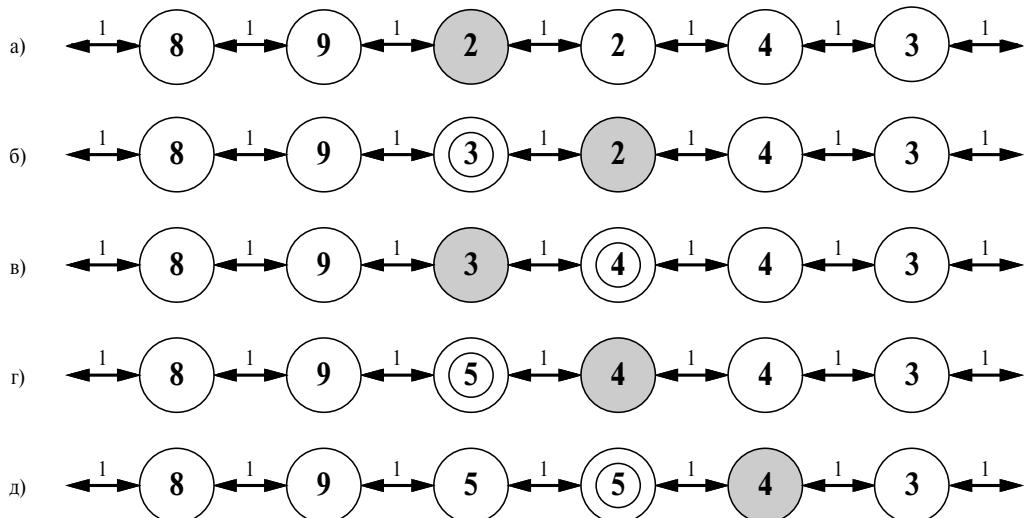


Рис. 4.15. Пять итераций алгоритма LRTA\* в одномерном пространстве состояний. Каждое состояние обозначено значением  $H(s)$ , текущей оценкой стоимости достижения цели, а каждая дуга обозначена соответствующей ей стоимостью этапа. Затененное состояние показывает местонахождение агента, а значения, обновленные после каждой итерации, обозначаются двойным кружком

Алгоритм агента, в котором реализована эта схема, получивший название поиска A\* в реальном времени с обучением (Learning Real-Time A\* —  $\text{LRTA}^*$ ), показан в листинге 4.6. Как и в алгоритме Online-DFS-Agent, в данном алгоритме составляется карта среды с использованием таблицы *result*. Этот алгоритм обновляет оценку стоимости для состояния, которое он только что оставил, а затем выбирает ход, “который представляется наилучшим” в соответствии с его текущими оценками стоимости. Следует отметить одну важную деталь, что действия, которые еще не были опробованы в состоянии  $s$ , всегда рассматриваются как ведущие непосредственно к цели с наименьшей возможной стоимостью, а именно  $h(s)$ . Такой **оптимизм в отношении неопределенности** побуждает агента исследовать новые пути, которые могут действительно оказаться перспективными.

**Листинг 4.6.** Функция **LRTA\*-Agent** выбирает действие в соответствии со значениями соседних состояний, которые обновляются по мере того, как агент передвигается в пространстве состояний

```
function LRTA*-Agent( $s'$ ) returns действие a
  inputs:  $s'$ , восприятие, позволяющее идентифицировать
          текущее состояние
```

```

static: result, таблица, индексированная по действиям и состояниям,
    первоначально пустая
    H, таблица оценок стоимостей, индексированная по состояниям,
    первоначально пустая
    s, a, предыдущие состояние и действие, первоначально
    неопределенные

if Goal-Test(s') then return stop
if s' является одним из новых состояний (отсутствующим в H)
    then H[s']  $\leftarrow h(s')$ 
unless s является неопределенным
    result[a, s]  $\leftarrow s'$ 

    H[s]  $\leftarrow \min_{b \in \text{Actions}(s)} \text{LRTA}^*-\text{Cost}(s, b, result[b, s], H)$ 

    a  $\leftarrow$  одно из действий b среди действий Actions(s'), которое
        минимизирует значение LRTA*-Cost(s', b, result[b, s'], H)
    s  $\leftarrow s'$ 
    return a

function LRTA*-Cost(s, a, s', H) returns оценка стоимости
    if s' является неопределенным then return h(s)
    else return c(s, a, s') + H[s']

```

---

Гарантируется, что агент LRTA<sup>\*</sup> найдет цель в любой конечной, безопасно исследуемой среде. Однако, в отличие от A<sup>\*</sup>, в бесконечных пространствах состояний применяемый в этом агенте алгоритм становится неполным, поскольку в некоторых случаях этот алгоритм может вовлечь агента в бесконечные блуждания. В наихудшем случае данный алгоритм позволяет исследовать среду с  $n$  состояниями за  $O(n^2)$  этапов, но чаще всего действует намного лучше. Агент LRTA<sup>\*</sup> представляет собой лишь один пример из большого семейства агентов, выполняющих поиск в оперативном режиме, которые могут быть определены путем задания правила выбора действия и правила обновления различными способами. Это семейство агентов, которое было первоначально разработано для стохастических вариантов среды, рассматривается в главе 21.

## Обучение в ходе поиска в оперативном режиме

То, что агенты, выполняющие поиск в оперативном режиме, на первых порах находятся в полном неведении, открывает некоторые возможности для обучения. Во-первых, агенты изучают “карту” своей среды (а точнее, результаты каждого действия в каждом состоянии), регистрируя результаты каждого из своих опытов. (Обратите внимание на то, что из предположения о детерминированности среды следует, что для изучения любого действия достаточно проведения одного эксперимента.) Во-вторых, агенты, выполняющие локальный поиск, получают все более точные оценки значения каждого состояния, используя локальные правила обновления, как в алгоритме LRTA<sup>\*</sup>. В главе 21 будет показано, что в конечном итоге такие обновления сходятся к точным значениям для каждого состояния, при условии, что агент исследует пространство состояний правильным способом. А после того как станут известными точные значения, оптимальные решения могут быть приняты путем перемещения к преемнику с наи-

высшим значением, т.е. в таком случае оптимальной стратегией становится метод поиска с восхождением к вершине в чистом виде.

Если читатель выполнил нашу рекомендацию провести трассировку поведения алгоритма Online-DFS-Agent в среде, показанной на рис. 4.12, то должен был заметить, что этот агент не слишком умен. Например, после того как агент обнаружил, что действие *Up* ведет из пункта  $(1, 1)$  в пункт  $(1, 2)$ , он еще не знает, что действие *Down* возвратит его назад в пункт  $(1, 1)$  или что следующее действие *Up* приведет его из пункта  $(2, 1)$  в пункт  $(2, 2)$ , из пункта  $(2, 2)$  в пункт  $(2, 3)$  и т.д. Вообще говоря, было бы желательно, чтобы агент освоил в результате обучения, что действие *Up* приводит к увеличению координаты  $y$ , если на этом пути нет стены, и что действие *Down* приводит к уменьшению этой координаты, и т.д. Для того чтобы это произошло, требуются две составляющие. Во-первых, необходимо формальное и явно манипулируемое представление общих правил такого рода; до сих пор мы скрывали эту информацию внутри “черного ящика”, называемого *функцией определения предметника*. Данному вопросу посвящена часть III. Во-вторых, нужны алгоритмы, позволяющие формировать подходящие общие правила из конкретных наблюдений, сделанных агентом. Эта тема рассматривается в главе 18.

## РЕЗЮМЕ

---

В данной главе рассматривалось применение **эвристических функций** для уменьшения стоимости поиска. В ней исследовался целый ряд алгоритмов, в которых используются эвристические функции, и было установлено, что даже при использовании хороших эвристических функций достижение оптимальности связано с увеличением стоимости поиска.

- **Поиск по первому наилучшему совпадению** сводится просто к применению алгоритма Graph-Search, в котором для развертывания выбираются неразвернутые узлы с минимальной стоимостью (в соответствии с некоторым критерием). В алгоритмах поиска по первому наилучшему совпадению обычно используется **эвристическая** функция  $h(n)$ , которая оценивает стоимость достижения решения из узла  $n$ .
- При **жадном поиске по первому наилучшему совпадению** развертываются узлы с минимальным значением  $h(n)$ . Этот поиск не оптимален, но часто является эффективным.
- При **поиске A\*** развертываются узлы с минимальным значением  $f(n) = g(n) + h(n)$ . Алгоритм A\* является полным и оптимальным, при условии, что гарантирована допустимость (для алгоритма Tree-Search) или преемственность (для алгоритма Graph-Search) функции  $h(n)$ . Пространственная сложность алгоритма A\* все еще остается слишком высокой.
- Производительность алгоритмов эвристического поиска зависит от качества эвристической функции. Иногда хорошие эвристические функции можно составить путем ослабления определения задачи, предварительного вычисления стоимости решения подзадач и сохранения этой информации в базе данных шаблонов или обучения на основе опыта решения данного класса задач.

- Алгоритмы **RBFS** и **SMA\*** представляют собой надежные, оптимальные алгоритмы поиска, в которых используются ограниченные объемы памяти; при наличии достаточного количества времени эти алгоритмы могут решать такие задачи, которые не позволяет решать алгоритм A\*, поскольку исчерпывает доступную память.
- Такие методы локального поиска, как **поиск с восхождением к вершине**, действуют на основе формулировок полного состояния и предусматривают хранение в памяти лишь небольшого количества узлов. Было разработано также несколько стохастических алгоритмов, включая поиск с **эмуляцией отжига**, которые возвращают оптимальные решения при наличии подходящего графика “охлаждения” (т.е. графика постепенного уменьшения величины случайного разброса). Кроме того, многие методы локального поиска могут использоваться для решения задач в непрерывных пространствах.
- **Генетический алгоритм** представляет собой стохастический поиск с восхождением к вершине, в котором сопровождается большая популяция состояний. Новые состояния формируются с помощью **мутации** и **скрещивания**; в последней операции комбинируются пары состояний, взятых из этой популяции.
- **Проблемы исследования** возникают, если агент не имеет никакого представления о том, каковы состояния и действия в его среде. Для безопасно исследуемых вариантов среди агенты, выполняющие **поиск в оперативном режиме**, могут составить карту и найти цель, если она существует. Эффективным методом выхода из локальных минимумов является обновление эвристических оценок на основе опыта.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Пример использования эвристической информации при решении задач впервые появился в одной из ранних статей Саймона и Ньюэлла [1419], но само выражение “евристический поиск” и обоснование применения эвристических функций, которые оценивают расстояние до цели, появились немного позже [932], [1126]. Доран и Мичи [404] осуществили обширные экспериментальные исследования в области применения эвристического поиска к решению ряда задач, в частности задач игры в восемь и игры в пятнадцать. Хотя Доран и Мичи провели теоретический анализ длины пути и “проникновения” (penetration) (отношения длины пути к общему количеству узлов, исследованных к данному моменту) в процессе эвристического поиска, они, по-видимому, игнорировали ту информацию, которую может предоставить текущая длина пути. Алгоритм A\*, предусматривающий использование в эвристическом поиске текущего значения длины пути, был разработан Хартом, Нильссоном и Рафаэлем [623], с некоторыми дальнейшими поправками [624]. Декстер и Перл [371] продемонстрировали оптимальную эффективность алгоритма A\*.

В указанной выше оригинальной статье, посвященной алгоритму A\*, было впервые представлено условие преемственности (consistency condition), которому должны удовлетворять эвристические функции. В качестве более простой замены этого условия Полом [1223] было предложено условие монотонности, но Перл [1188] показал, что эти два условия эквивалентны. Аналоги открытых и закрытых списков использовались

во многих алгоритмах, явившихся предшественниками алгоритма A\*, к ним относятся алгоритмы поиска в ширину, поиска в глубину и поиска по критерию стоимости [97], [399]. Важность применения аддитивных стоимостей путей для упрощения алгоритмов оптимизации особо ярко была показана в работе Беллмана [97].

Пол [1220], [1223] впервые провел исследования связи между ошибками в эвристических функциях и временной сложностью алгоритма A\*. Доказательство того, что работа алгоритма A\* завершается за линейное время, если ошибка в эвристической функции ограничена некоторой константой, можно найти в работах Пола [1223] и Гашнига [522]. Перл [1188] развил этот результат, что позволило учитывать логарифмический рост ошибки. Применение “эффективного коэффициента ветвлений” в качестве критерия эффективности эвристического поиска было предложено Нильссоном [1141].

Существует много вариантов алгоритма A\*. Пол [1222] предложил использовать в алгоритме A\* динамическое взвешивание (dynamic weighting), в котором в качестве функции оценки применяется взвешенная сумма текущей длины пути и эвристической функции  $f_w(n) = w_g g(n) + w_h h(n)$ , а не просто сумма  $f(n) = g(n) + h(n)$ . Веса  $w_g$  и  $w_h$  корректируются динамически по мере развития поиска. Можно доказать, что алгоритм Пола является  $\epsilon$ -допустимым (т.е. гарантирует нахождение решений с отклонением от оптимального решения на коэффициент  $1+\epsilon$ ), где  $\epsilon$  — параметр, предусмотренный в алгоритме. Таким же свойством обладает алгоритм  $A_\epsilon^*$  [1188], который способен выбрать из периферии любой узел, при условии, что его f-стоимость находится в пределах коэффициента  $1+\epsilon$  от стоимости периферийного узла с наименьшей f-стоимостью. Выбор соответствующего коэффициента может быть сделан таким образом, чтобы существовала возможность минимизировать стоимость поиска.

Алгоритм A\* и другие алгоритмы поиска в пространстве состояний тесно связаны с методами ветвей и границ, которые широко используются в исследованиях операций [901]. Соотношения между алгоритмами поиска в пространстве состояний и методами ветвей и границ были глубоко исследованы в [867], [869], [1115]. Мартелли и Монтанари [990] показали связь между динамическим программированием (см. главу 17) и некоторыми типами поиска в пространстве состояний. Кумар и Канал [868] предприняли попытку “великой унификации” методов эвристического поиска, динамического программирования, а также методов ветвей и границ под общим названием CDP (Composite Decision Process — комплексный процесс принятия решений).

Поскольку компьютеры в конце 1950-х — начале 1960-х годов имели не больше нескольких тысяч слов оперативной памяти, темой ранних исследовательских работ часто служил эвристический поиск с ограничением памяти. Одна из самых первых программ поиска, Graph Traverser [404], фиксирует свои результаты в виде некоторого оператора после выполнения поиска по первому наилучшему совпадению вплоть до заданного предела объема памяти. Алгоритм IDA\* [835], [836] стал одним из первых широко применяемых оптимальных алгоритмов эвристического поиска с ограничением памяти, после чего было разработано большое количество его вариантов. Анализ эффективности алгоритма IDA\* и сложностей, возникающих при его применении с эвристическими функциями, которые встречаются в реальных задачах, приведен в работе Патрика и др. [1182].

Алгоритм RBFS [840], [841] фактически является лишь немного более сложным по сравнению с алгоритмом, приведенным в листинге 4.1. Последний вариант ближе к независимо разработанному алгоритму, получившему название алгоритма **итеративного развертывания**, или сокращенно IE (Iterative Expansion) [1324]. В алгоритме RBFS используется не только верхняя, но и нижняя граница; эти два алгоритма при использовании допустимых эвристических функций ведут себя одинаково, но RBFS развертывает узлы в порядке первого наилучшего совпадения даже при наличии недопустимой эвристической функции. Идея отслеживания наилучшего альтернативного пути появилась еще раньше в изящной реализации алгоритма A\* на языке Prolog, предложенной Братко [174], и в алгоритме DTA\* [1332]. В последней работе обсуждаются также метауровневые пространства состояний и метауровневое обучение.

Алгоритм MA\* впервые опубликован в [229]. Появление алгоритма SMA\*, или Simplified MA\*, стало результатом попытки реализовать MA\* в качестве алгоритма сравнения для метода IE [1324]. Кайндл и Корсанд [763] применили алгоритм SMA\* для создания алгоритма двунаправленного поиска, который функционирует значительно быстрее по сравнению с предшествующими алгоритмами. В [845] описан подход по принципу “разделяй и властвуй”, а в [1645] представлен алгоритм A\* поиска в графе с ограничением памяти. Обзор методов поиска с ограничением памяти приведен в [842].

Идея о том, что допустимые эвристические функции могут быть получены с помощью ослабления условий задачи, впервые опубликована в оригинальной статье [645], авторы которой использовали эвристику на основе минимального связующего дерева для решения задачи TSP (см. упр. 4.8).

Автоматизация процесса ослабления условий задачи была успешно осуществлена Придитисом [1237] на основе его более ранней совместной работы с Мостоу [1092]. Использование баз данных шаблонов для составления допустимых эвристических функций предложено в [313] и [523]; базы данных с непересекающимися шаблонами описаны в [844]. Вероятностная интерпретация эвристических функций глубоко исследована в [616] и [1188].

Безусловно, наиболее исчерпывающим источником сведений об эвристических функциях и алгоритмах эвристического поиска является книга Перла *Heuristics* [1188]. В этой книге приведено особенно хорошее описание широкого разнообразия версий и вариантов алгоритма A\*, включая строгие доказательства их формальных свойств. В работе Канала и Кумара представлена антология важных статей по эвристическому поиску [767]. Результаты новейших исследований в области алгоритмов поиска регулярно публикуются в журнале *Artificial Intelligence*.

Методы локального поиска имеют долгую историю в математике и компьютерных науках. Безусловно, как очень эффективный метод локального поиска в непрерывных пространствах, в которых доступна информация о градиенте, может рассматриваться метод Ньютона–Рафсона [1132], [1266]. В [182] можно найти классический справочник по алгоритмам оптимизации, для которых не требуется такая информация. Алгоритм лучевого поиска, представленный в данной книге как алгоритм локального поиска, впервые появился в виде одного из вариантов методов динамического программирования с ограничением по ширине, предназначенного для распознавания речи, в системе Наргу [952]. Еще один алгоритм подобного типа был глубоко проанализирован Перлом [1188] (глава 5).

В последние годы снова пробудился интерес к теме локального поиска под влиянием исключительно качественных результатов, полученных при решении таких крупных задач удовлетворения ограничений, как задача с  $n$ -ферзями [1058] и задача формирования логических рассуждений [1382], а также под влиянием успешного включения в алгоритмы методов рандомизации, методов множественного одновременного поиска и других усовершенствований. Это возрождение интереса к алгоритмам, названным Кристосом Пападимитриу алгоритмами “новой эры”, вызвало также повышенный интерес теоретиков в области компьютерных наук [13], [848]. В области исследования операций завоевал широкую популярность один из вариантов поиска с восхождением к вершине, получивший название **поиска с запретами** [563], [564]. В этом алгоритме, основанном на моделях ограниченной кратковременной памяти человека, ведется список запретов, состоящий из  $k$  ранее посещенных состояний, которые нельзя посещать повторно; это позволяет данному алгоритму не только выполнять поиск в графах более эффективно, но и выходить из некоторых локальных минимумов. Еще одним полезным усовершенствованием алгоритма поиска с восхождением к вершине является алгоритм Stage [163]. Идея этого алгоритма состоит в том, что для получения представления об общей форме ландшафта должны использоваться локальные максимумы, найденные в результате восхождения к вершине с перезапуском случайнym образом. Данный алгоритм согласует гладкую поверхность с множеством локальных максимумов, а затем аналитическим путем вычисляет глобальный максимум этой поверхности. Полученный глобальный максимум становится новой точкой перезапуска. Успешное функционирование этого алгоритма было испытано на практике при решении трудных задач. В [573] показано, что распределения времени выполнения алгоритмов, в которых систематически применяются возвраты, часто имеют форму **распределений с широким хвостом** (heavy-tailed distribution), а это означает, что вероятность очень продолжительного времени выполнения выше, чем можно было бы предположить в случае нормального распределения значений времени выполнения. Данные результаты могут служить теоретическим обоснованием применимости алгоритмов с перезапуском случайнym образом.

Метод поиска с эмуляцией отжига был впервые описан Кирпатриком и др. [799], которые заимствовали соответствующую идею непосредственно из **алгоритма Метрополиса** (этот алгоритм применялся для эмуляции сложных систем в физике [1036] и был, как многие полагают, изобретен на одном из званых обедов в Лос-Аламосе). Методы поиска с эмуляцией отжига теперь лежат в основе отдельного научного направления, в котором ежегодно публикуются сотни статей.

Поиск оптимальных решений в непрерывных пространствах является предметом исследований в нескольких научных областях, включая **теорию оптимизации, теорию оптимального управления и вариационное исчисление**. Подходящие для нашей темы (и практически применимые) вступительные сведения приведены в [133] и [1236]. Одним из первых приложений для компьютеров было **линейное программирование** (Linear Programming — LP); относящийся к этой области **симплексный алгоритм** [322], [1611] все еще используется, несмотря на то, что в наихудшем случае он характеризуется экспоненциальной сложностью. Кармаркар [771] разработал практически применимый алгоритм для задач LP с полиномиальной временной сложностью.

Важной работой, предшествующей разработке генетических алгоритмов, было исследование концепции **ландшафта пригодности** (fitness landscape), проведенное Сьюэллом Райтом [1623]. В 1950-х годах некоторые специалисты в области стати-

стики, включая Бокса [161] и Фридмана [504], использовали эволюционные методы для решения задач оптимизации, но этот подход приобрел популярность только после того, как Рехенберг [1270], [1271] предложил использовать **стратегии эволюции** (evolution strategy) для решения задач оптимизации аэродинамических поверхностей. В 1960-х и 1970-х годах генетические алгоритмы глубоко исследовал Джон Холланд [669], применяя их и в качестве полезного инструментального средства, и в качестве способа расширения знаний в области адаптации, как биологической, так и связанной с другими научными направлениями [670]. Сторонники движения, доказывающие возможность **искусственной жизни** [886], развили эту идею на один шаг дальше, рассматривая продукты генетических алгоритмов как организмы, а не как решения задач. В результате исследований в этой области, проведенных Хинтоном и Ноуланом [656], а также Экли и Литтманом [3], было сделано очень многое для выяснения последствий действия эффекта Болдуина. Для ознакомления с общими сведениями об эволюции авторы настоятельно рекомендуют книгу [1439].

Сравнение генетических алгоритмов с другими подходами (особенно с алгоритмом стохастического поиска с восхождением к вершине) чаще всего показывает, что генетические алгоритмы сходятся более медленно [66], [754], [1060], [1158]. Такие исследования не находят всеобщего признания в сообществе приверженцев алгоритмов GA, но недавние попытки представителей этого сообщества трактовать поиск на основе популяций как приближенный аналог байесовского обучения (см. главу 20) могут способствовать преодолению разногласий между представителями этой области и ее критиками [1200]. Для анализа производительности алгоритмов GA может также применяться теория **квадратичных динамических систем** [1262]. В [944] можно найти пример применения GA для проектирования антенн, а в [890] — пример применения этих алгоритмов для решения задачи коммивояжера.

С генетическими алгоритмами тесно связана область **генетического программирования**. Принципиальное различие между ними состоит в том, что представлениями, к которым применяются операции мутации и комбинирования, являются программы, а не битовые строки. Программы представлены в форме деревьев выражений; выражения могут быть сформированы на таком стандартном языке, как Lisp, или специально спроектированы для представления технических схем, контроллеров роботов и т.д. Операция скрещивания предусматривает соединение друг с другом поддеревьев, а не подстрок. Такая форма мутации гарантирует, что потомки будут представлять собой правильно построенные выражения, а это не было бы возможно в случае проведения манипуляций с программами, представленными в виде строк.

Недавно возникший широкий интерес к генетическому программированию был стимулирован работой Джона Козы [855], [856], но исследования в этой области проводились уже давно, начиная с экспериментов с машинным кодом [502] и с конечными автоматами [476]. Как и в отношении генетических алгоритмов, еще не утихли споры по поводу эффективности методов генетического программирования. В [857] описаны результаты ряда экспериментов по автоматизированному проектированию схемотехнических устройств с использованием генетического программирования.

Работы в области генетических алгоритмов и генетического программирования публикуются в журналах *Evolutionary Computation* и *IEEE Transactions on Evolutionary Computation*; статьи на эти темы можно также найти в журналах *Complex Systems*, *Adaptive Behavior* и *Artificial Life*. Основными конференциями являются *International Conference on Genetic Algorithms* и *Conference on Genetic Programming*, а недавно произошло их слияние и создана конференция *Genetic and Evolutionary Computation*.

*Conference*. Хорошие обзоры этой области приведены в книгах Мелани Митчелл [1059] и Дэвида Фогеля [475].

Алгоритмы исследования неизвестных пространств состояний привлекали интерес ученых в течение многих столетий. Поиск в глубину в лабиринте можно осуществить, постоянно держась левой рукой за стену; циклов можно избежать, отмечая каждую развилку. При наличии необратимых действий поиск в глубину оканчивается неудачей; более общая задача исследования **графов Эйлера** (т.е. графов, в которых каждый узел имеет одинаковое число входящих и исходящих ребер) была решена с помощью алгоритма, предложенного Хирхольцером [652]. Первое исчерпывающее описание алгоритмов решения задачи исследования для произвольных графов было приведено в [385]; авторы этой работы предложили полностью общий алгоритм, но показали, что невозможно определить ограниченный коэффициент конкурентоспособности для задачи исследования графа общего вида. В [1171] приведены результаты исследования проблемы поиска путей к цели в геометрических вариантах среды планирования пути (где все действия являются обратимыми). Эти результаты показали, что достичь того, чтобы коэффициент конкурентоспособности оставался небольшим, можно при наличии квадратных препятствий, но если препятствия имеют произвольную прямоугольную форму, то невозможно добиться того, чтобы значения коэффициента конкурентоспособности оставались ограниченными (см. рис. 4.13).

Алгоритм LRTA\* был разработан Корфом [839] в ходе исследований в области **поиска в реальном времени** для вариантов среды, в которых агент должен действовать после проведения поиска лишь в течение ограниченного времени (этота ситуация встречается гораздо чаще в играх с двумя участниками). По сути, алгоритм LRTA\* представляет собой частный случай алгоритмов обучения с подкреплением для стохастических вариантов среды [74]. Применяемый в нем принцип оптимистического отношения к неопределенности (согласно которому следует всегда отправляться к ближайшему непосещенному состоянию) может в случае отсутствия информации о среде привести к формированию такого подхода к исследованию, который является менее эффективным по сравнению с простым поиском в глубину [818]. В [327] показано, что поиск с итеративным углублением в оперативном режиме обладает оптимальной эффективностью поиска цели в однородном дереве при отсутствии эвристической информации. В сочетании с различными методами поиска и обновления в известной части графа было разработано несколько вариантов алгоритма LRTA\*, предусматривающих использование информации о среде [1201]. Тем не менее еще нет полного понимания того, как следует находить цели с оптимальной эффективностью при использовании эвристической информации.

Тема алгоритмов **параллельного поиска** в этой главе не рассматривалась, отчасти потому, что для этого требуется привести подробное описание параллельных компьютерных архитектур. Параллельный поиск становится важной темой и в искусственном интеллекте, и в теоретических компьютерных науках. Краткое введение в тематику литературы по искусственноому интеллекту можно найти в книге Маханти и Дэниелса [969].

## УПРАЖНЕНИЯ

- 4.1.** Выполните трассировку работы алгоритма поиска A\* применительно к решению задачи проезда в город Бухарест из города Лугож с использованием эвристической функции определения расстояния по прямой. Иными словами, покажите последовательность узлов, которые рассматриваются этим алгоритмом, и приведите оценки  $f$ ,  $g$  и  $h$  для каждого узла.
- 4.2.** Эвристический алгоритм поиска пути представляет собой алгоритм поиска по первому наилучшему совпадению, в котором целевая функция равна  $f(n) = (2-w)g(n) + wh(n)$ . Для каких значений  $w$  гарантируется оптимальность этого алгоритма? (Может быть принято предположение, что функция  $h$  является допустимой.) Какого рода поиск выполняется в этом алгоритме, когда  $w=0$ ? Когда  $w=1$ ? Когда  $w=2$ ?
- 4.3.** Докажите каждое из приведенных ниже утверждений.
- Поиск в ширину представляет собой частный случай поиска по критерию стоимости.
  - Поиск в ширину, поиск в глубину и поиск по критерию стоимости (uniform-cost search) — специальные случаи поиска по первому наилучшему совпадению.
  - Поиск по критерию стоимости — специальный случай поиска A\*.
- 4.4.**  Предложите такое пространство состояний, в котором применение поиска A\* с использованием алгоритма Graph-Search приводит к получению неоптимального решения при наличии функции  $h(n)$ , которая является допустимой, но непреемственной.
- 4.5.** На с. 156 было показано, что применение эвристической функции определения расстояния по прямой приводит к тому, что жадный поиск по первому наилучшему совпадению безвозвратно углубляется в дерево поиска при решении задачи проезда от города Яссы до города Фэгэраш. Однако эта эвристическая функция работает идеально при решении противоположной задачи — поиска пути проезда от города Фэгэраш до города Яссы. Существуют ли такие задачи, в которых эта эвристическая функция исключает возможность успешного выполнения задачи поиска пути в любом из направлений?
- 4.6.** Предложите эвристическую функцию для задачи игры в восемь, которая иногда допускает переоценку, и покажите, каким образом это может привести к получению неоптимального решения некоторых конкретных экземпляров задачи. (Для упрощения работы при выполнении упражнения можете использовать компьютер.) Докажите, что если функция  $h$  никогда не переоценивает стоимость больше чем на  $c$ , то алгоритм A\* с использованием функции  $h$  возвращает решение, стоимость которого превышает оптимальное решение не больше чем на  $c$ .
- 4.7.** Докажите, что если эвристическая функция является преемственной, то она должна быть допустимой. Составьте допустимую эвристическую функцию, которая не является преемственной.

- 4.8.** Задача коммивояжера (Traveling Salesperson Problem — TSP) может быть решена с помощью эвристической функции, основанной на определении минимального связующего дерева (Minimum Spanning Tree — MST), которая используется для оценки стоимости завершения обхода, при условии, что уже был сформирован частичный путь обхода. Стоимость MST для множества городов представляет собой минимальную сумму стоимостей соединений в любом дереве, которое связывает все города.
- Покажите, как можно вывести эту эвристическую функцию на основе одной из ослабленных версий задачи TSP.
  - Покажите, что эвристическая функция MST доминирует над эвристической функцией определения расстояния по прямой.
  - Напишите генератор задач для создания экземпляров задачи TSP, в которых города представлены случайно выбранными точками в единичном квадрате.
  - Найдите в литературе эффективный алгоритм формирования MST и примените его в сочетании с допустимым алгоритмом поиска для решения экземпляров задачи TSP.
- 4.9.** На с. 171 был определен ослабленный вариант задачи игры в восемь, в котором любая фишка может перемещаться из квадрата  $A$  в квадрат  $B$ , если квадрат  $B$  является пустым. Точное решение этой задачи предусматривает определение **эвристической функции Гашнига** [522]. Объясните, почему эвристическая функция Гашнига является, по меньшей мере, такой же точной, как и функция  $h_1$  (в которой учитываются фишки, стоящие не на месте), и продемонстрируйте случай, в которых она является более точной по сравнению и с  $h_1$ , и с  $h_2$  (в которой используется манхэттенское расстояние). Можете ли вы предложить способ эффективного вычисления эвристической функции Гашнига?
- 4.10.** В данной главе были описаны две простые эвристические функции для задачи игры в восемь; в одной из них учитывается манхэттенское расстояние, а в другой — стоящие не на месте фишки. В литературе предложено также несколько других эвристических функций, предназначенных для использования в качестве улучшенной замены этих двух (см., например, [617], [1092] и [1141]). Прoverьте обоснованность этих претензий, реализовав соответствующие эвристические функции и сравнив производительность полученных алгоритмов.
- 4.11.** Укажите название алгоритма, который фактически становится следствием применения каждого из перечисленных ниже частных случаев.
- Локальный лучевой поиск с  $k=1$ .
  - Локальный лучевой поиск с одним начальным состоянием и без ограничений на количество хранимых состояний.
  - Моделируемый отжиг с постоянным значением  $T=0$  (и исключенной проверкой завершения).
  - Генетический алгоритм с размером популяции  $N=1$ .
- 4.12.** Иногда не существует хорошей функции оценки для некоторой задачи, но имеется хороший метод сравнения — способ определения того, является ли один узел лучше другого, без присваивания числовых значений тому и друго-

- му. Покажите, что этого достаточно для выполнения поиска по первому наилучшему совпадению. Есть ли соответствующий аналог алгоритма A\*?
- 4.13. Определите, как связана временная сложность алгоритма LRTA\* с его пространственной сложностью.
- 4.14. Предположим, что некоторый агент находится в среде лабиринта  $3 \times 3$ , подобного приведенному на рис. 4.12. Агенту известно, что его первоначальным местонахождением является пункт  $(1, 1)$ , что цель находится в пункте  $(3, 3)$  и что четыре действия, *Up*, *Down*, *Left*, *Right*, приводят к своим обычным последствиям, если не будут заблокированы стеной. Агент не знает о том, где находятся внутренние стены. В любом конкретном состоянии агент воспринимает информацию о множестве допустимых действий; он также может определить, является ли данное состояние тем, которое он уже посещал ранее, или представляет собой новое состояние.
- Объясните, каким образом к этой задаче поиска в оперативном режиме можно применить такую трактовку, чтобы она могла рассматриваться как поиск в автономном режиме в пространстве доверительных состояний, где первоначальное доверительное состояние включает все возможные конфигурации среды. Насколько велико это первоначальное доверительное состояние? Насколько велик все пространство доверительных состояний?
  - Какое количество различных вариантов восприятия возможно в первоначальном состоянии?
  - Опишите первые несколько ветвей плана действий в непредвиденных ситуациях для этой задачи. Насколько велик этот план в полном виде (дайте приблизительную оценку)?
- Обратите внимание на то, что этот план действий в непредвиденных ситуациях является решением для каждой возможной среды, соответствующей данному описанию. Поэтому, строго говоря, чередование поиска и выполнения не требуется даже в неизвестных вариантах среды.
- 4.15. В данном упражнении исследуется направление использования локальных методов поиска для решения задач TSP такого типа, который определен в упр. 4.8.
- Предложите подход к решению задач TSP на основе поиска с восхождением к вершине. Сравните эти результаты с оптимальными решениями, полученными на основе алгоритма A\* с эвристической функцией MST (упр. 4.8).
  - Предложите подход к решению задачи коммивояжера с помощью генетического алгоритма. Сравните результаты обоих подходов. Для ознакомления с некоторыми рекомендациями по выбору представлений вы можете обратиться к [890].
- 4.16. Сформируйте большое количество экземпляров игры в восемь и задачи с восемью ферзями и найдите их решения (там, где это возможно) с помощью поиска с восхождением к вершине (в варианте подъема по самому крутым склону и в варианте поиска по первому наилучшему совпадению), поиска с восхождением к вершине с перезапуском случайным образом и поиска с эмуляцией отжига. Измерьте стоимость поиска и процентное соотношение решенных за-

дач, а также нанесите эти данные на график наряду с данными об оптимальной стоимости решения. Прокомментируйте полученные вами результаты.

- 4.17. В данном упражнении исследуется применение поиска с восхождением к вершине в контексте робототехнической навигации, с использованием в качестве примера среды, показанной на рис. 3.14.
- а) Повторите упр. 3.16 с использованием поиска с восхождением к вершине. Зашел ли когда-либо предложенный вами агент в тупик в локальном минимуме? Возможно ли, чтобы он зашел в тупик при наличии выпуклых препятствий?
  - б) Сформируйте среду с препятствиями в виде невыпуклых многоугольников, в которой агент может оказаться в тупике.
  - в) Модифицируйте алгоритм поиска с восхождением к вершине таким образом, чтобы вместо выполнения поиска на глубину 1 для принятия решения о том, куда двигаться дальше, агент предпринимал бы поиск на глубину  $k$ . Он должен находить наилучший путь, состоящий из  $k$  этапов, проходить по нему на один этап, а затем повторять процесс.
  - г) Существует ли такое значение  $k$ , при котором новый алгоритм гарантирует выход из локальных минимумов?
  - д) Объясните, благодаря чему алгоритм LRTA\* позволяет агенту выходить из локальных минимумов в этом случае.
- 4.18. Сравните производительность алгоритмов A\* и RBFS на множестве случайно сформированных экземпляров задач в проблемных областях задачи игры в восемь (с манхэттенским расстоянием) и задачи TSP (с MST — см. упр. 4.8). Обсудите полученные вами результаты. Как изменяется производительность алгоритма RBFS после добавления небольшого случайного числа к эвристическим значениям в проблемной области задачи игры в восемь?

# 5 ЗАДАЧИ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ

*В этой главе показано, что, рассматривая состояния на более высоком уровне детализации, чем просто как маленькие “черные ящики”, можно прийти к созданию целого ряда мощных новых методов поиска и более глубокому пониманию структуры и сложности задачи.*

В главах 3 и 4 рассматривался подход, согласно которому задачи можно решать, выполняя поиск в пространстве **состояний**. Для реализации этого подхода состояния необходимо оценивать с помощью эвристических функций, соответствующих данной проблемной области, а также проверять для определения того, не являются ли они целевыми состояниями. Однако с точки зрения таких алгоритмов поиска каждое состояние представляет собой  **черный ящик** с неразличимой внутренней структурой. Они представлены с помощью произвольно выбранной структуры данных, доступ к которой может осуществляться только с применением процедур, относящихся к данной проблемной области, — функции определения преемника, эвристической функции и процедуры проверки цели.

В настоящей главе рассматриваются **задачи удовлетворения ограничений**, в которых состояния и проверка цели соответствуют стандартному, структурированному и очень простому  **представлению** (см. раздел 5.1). Это позволяет определять алгоритмы поиска, способные воспользоваться преимуществом такой структуры состояний, и применять для решения крупных задач эвристические функции общего назначения, а не функции, относящиеся к конкретной задаче (см. разделы 5.2, 5.3). Но, возможно, еще важно то, что применяемое стандартное представление проверки цели раскрывает структуру самой задачи (см. раздел 5.4). Это позволяет создать методы декомпозиции задачи и понять внутреннюю связь между структурой задачи и сложностью ее решения.

## 5.1. ЗАДАЧИ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ

Формально говоря, **любая задача удовлетворения ограничений** (Constraint Satisfaction Problem — CSP) определена множеством **переменных**,  $X_1, X_2, \dots, X_n$ , и множеством **ограничений**,  $C_1, C_2, \dots, C_m$ . Каждая переменная  $X_i$  имеет непустую **область определения**  $D_i$  возможных **значений**. Каждое ограничение  $C_i$  включает некоторое подмножество переменных и задает допустимые комбинации значений для этого подмножества. Состояние задачи определяется путем **присваивания** значений некоторым или всем этим переменным,  $\{X_i = v_i, X_j = v_j, \dots\}$ . Присваивание, которое не нарушает никаких ограничений, называется **совместимым**, или допустимым присваиванием. Полным называется такое присваивание, в котором участвует каждая переменная, а **решением** задачи CSP является полное присваивание, которое удовлетворяет всем ограничениям. Кроме того, для некоторых задач CSP требуется найти решение, которое максимизирует **целевую функцию**.

Как же все это описание перевести на язык практики? Предположим, что устав от Румынии, мы рассматриваем карту Австралии, на которой показаны все ее штаты и территории (рис. 5.1, *a*), и что мы получили задание раскрасить каждый регион в красный, зеленый или синий цвет таким способом, чтобы ни одна пара соседних регионов не имела одинаковый цвет. Чтобы сформулировать это задание в виде задачи CSP, определим в качестве переменных сокращенные обозначения этих регионов: *WA*, *NT*, *Q*, *NSW*, *V*, *SA* и *T*. Областью определения каждой переменной является множество цветов  $\{\text{red}, \text{green}, \text{blue}\}$ . Ограничения требуют, чтобы все пары соседних регионов имели разные цвета; например, допустимыми комбинациями для *WA* и *NT* являются следующие пары:

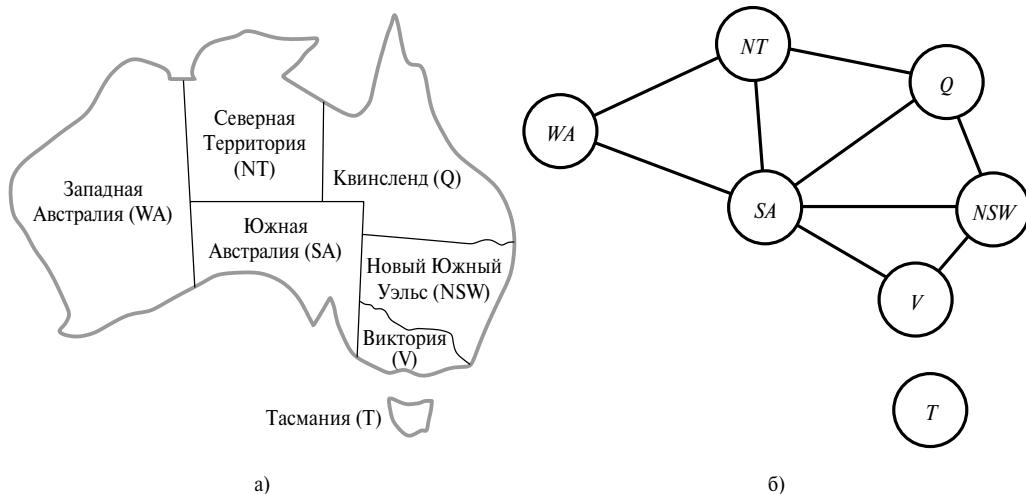
```
{(red, green), (red, blue), (green, red), (green, blue), (blue, red),
 (blue, green)}
```

(Это ограничение можно также представить более сжато в виде неравенства  $WA \neq NT$ , при условии, что в данном алгоритме удовлетворения ограничений предусмотрен некоторый способ вычисления таких выражений.) Количество возможных решений достаточно велико; например, одним из таких решений является следующее:

```
{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=red}
```

Задачу CSP удобно представить визуально в виде **графа ограничений**, как показано на рис. 5.1, *b*. Узлы этого графа соответствуют переменным задачи, а дуги — ограничениям.

Рассматривая некоторую задачу в виде задачи CSP, можно достичь нескольких важных преимуществ. Представление состояний в задаче CSP соответствует некоторому стандартному шаблону (т.е. выражается в виде множества переменных с присвоенными значениями), поэтому функцию определения преемника и проверку цели можно записать в универсальной форме, применимой ко всем задачам CSP. Более того, могут быть разработаны эффективные, универсальные эвристические функции, для создания которых не требуются дополнительные знания о конкретной проблемной области. Наконец, для упрощения процесса решения может использоваться сама структура графа ограничений, что позволяет в некоторых случаях добиться экспоненциального уменьшения сложности. Это представление задачи CSP является первым и простейшим из ряда схем представления, которые будут разрабатываться на протяжении данной книги.



*Рис. 5.1. Пример определения задачи CSP: основные штаты и территории на карте Австралии (а); раскраска этой карты может рассматриваться как задача удовлетворения ограничений; задание состоит в том, чтобы назначить цвет каждому региону так, что ни одна пара соседних регионов не будет иметь одинаковый цвет; рассматриваемая задача раскраски карты, представленная в виде графа ограничений (б)*

Можно довольно легко показать, что любой задаче CSP может быть дана **инкрементная формулировка**, как и любой стандартной задаче поиска, следующим образом.

- **Начальное состояние.** Пустое присваивание {}, в котором ни одной переменной не присвоено значение.
- **Функция определения преемника.** Значение может быть присвоено любой переменной с неприсвоенным значением, при условии, что переменная не будет конфликтовать с другими переменными, значения которым были присвоены ранее.
- **Проверка цели.** Текущее присваивание является полным.
- **Стоимость пути.** Постоянная стоимость (например, 1) для каждого этапа.

Каждое решение должно представлять собой полное присваивание и поэтому должно находиться на глубине  $n$ , если имеется  $n$  переменных. Кроме того, дерево поиска распространяется только на глубину  $n$ . По этим причинам для решения задач CSP широко применяются алгоритмы поиска в глубину (см. раздел 5.2). К тому же такие задачи характерны тем, что ~~сам путь, по которому достигается некоторое решение, не представляет интереса~~. Поэтому может также использоваться **формулировка с полным состоянием**, в которой каждое состояние представляет собой полное присваивание, удовлетворяющее или не удовлетворяющее ограничениям. На основе такой формулировки хорошо действуют методы локального поиска (см. раздел 5.3).

Задачи CSP простейшего вида характеризуются тем, что в них используются переменные, которые являются **дискретными** и имеют ~~конечные области определения~~. К такому виду относится задача раскраски карты. Задача с восемью

ферзями, описанная в главе 3, также может рассматриваться как задача CSP с конечной областью определения, где переменные  $Q_1, \dots, Q_8$  представляют собой позиции каждого ферзя на столбцах 1, ..., 8, а каждая переменная имеет область определения  $\{1, 2, 3, 4, 5, 6, 7, 8\}$ . Если максимальный размер области определения любой переменной в задаче CSP равен  $d$ , то количество возможных полных присваиваний измеряется величиной  $O(d^n)$ , т.е. зависит экспоненциально от количества переменных. К категории задач CSP с конечной областью определения относятся **булевы задачи CSP**, в которых переменные могут иметь значения либо *true*, либо *false*. Булевы задачи CSP включают в качестве частных случаев некоторые NP-полные задачи, такие как 3SAT (см. главу 7). Поэтому в худшем случае нельзя рассчитывать на то, что мы сможем решать задачи CSP с конечной областью определения за время меньше экспоненциального. Но в большинстве практических приложений алгоритмы CSP общего назначения позволяют решать задачи, на несколько порядков величины более крупные по сравнению с теми, которые могут быть решены с помощью алгоритмов поиска общего назначения, представленных в главе 3.

Дискретные переменные могут также иметь **бесконечные области определения**, например, такие, как множество всех целых чисел или множество всех строк. В частности, при календарном планировании строительных работ дата начала каждой работы представляет собой переменную, а ее возможными значениями являются целочисленные интервалы времени в сутках, отсчитываемые от текущей даты. При решении задач с бесконечными областями определения больше нет возможности описывать ограничения, перечисляя все допустимые комбинации значений. Вместо этого должен использоваться **язык ограничений**. Например, если работа  $Job_1$ , которая занимает пять дней, должна предшествовать работе  $Job_3$ , то для описания этого условия потребуется язык ограничений, представленных в виде алгебраических неравенств, таких как  $StartJob_1 + 5 \leq StartJob_3$ . Кроме того, больше нет возможности решать такие ограничения, просто перечисляя все возможные присваивания, поскольку количество подобных возможных присваиваний бесконечно велико. Для **линейных ограничений** с целочисленными переменными (т.е. ограничений, подобных только что приведенному, в которых каждая переменная представлена только в линейной форме) существуют специальные алгоритмы поиска решений (которые здесь не рассматриваются). Можно доказать, что не существует алгоритмов решения общих **нелинейных ограничений** с целочисленными переменными. В некоторых случаях задачи с целочисленными ограничениями можно свести к задачам с конечной областью определения, устанавливая пределы значений всех этих переменных. Например, в задаче планирования можно установить верхний предел, равный общей продолжительности всех работ, которые должны быть запланированы.

В реальном мире очень часто встречаются задачи удовлетворения ограничений с **непрерывными областями определения**, и эти задачи интенсивно изучаются в области исследования операций. Например, для планирования экспериментов на космическом телескопе Хаббл требуется очень точная привязка наблюдений по времени; начало и конец каждого наблюдения и каждого маневра представляют собой переменные со значениями из непрерывной области опре-

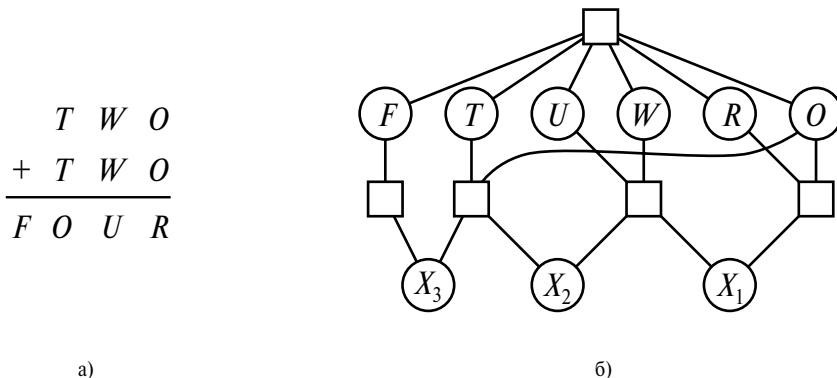
деления, которые должны подчиняться всевозможным астрономическим ограничениям, ограничениям предшествования и ограничениям по мощности двигателей. Одной из широко известных категорий задач CSP с непрерывной областью определения являются задачи **линейного программирования**, в которых ограничения должны представлять собой линейные неравенства, образующие выпуклую область. Задачи линейного программирования могут быть решены за время, которое зависит полиномиально от количества переменных. Кроме того, проводились исследования задач с другими типами ограничений и целевых функций — задачи квадратичного программирования, конического программирования второго порядка и т.д.

Кроме исследования типов переменных, которые могут присутствовать в задачах CSP, полезно заняться изучением типов ограничений. Простейшим типом ограничения является **унарное ограничение**, которое ограничивает значение единственной переменной. Например, может оказаться, что жители штата Южная Австралия очень не любят зеленый цвет, {green}. Каждое унарное ограничение можно устраниТЬ, выполняя предварительную обработку области определения соответствующей переменной, чтобы удалить любое значение, нарушающее это ограничение. **Бинарное ограничение** связывает между собой две переменные. Например, бинарным ограничением является  $SA \neq NSW$ . Бинарной задачей CSP называется задача, в которой предусмотрены только бинарные ограничения; она может быть представлена в виде графа ограничений, подобного приведенному на рис. 5.1, б.

В ограничениях высокого порядка участвуют три или больше переменных. Одним из известных примеров таких задач являются **криптоарифметические головоломки**, называемые также **числовыми ребусами** (рис. 5.2, а). Обычно предъявляется требование, чтобы каждая буква в криптоарифметической головоломке представляла отдельную цифру. В случае задачи, показанной на рис. 5.2, а, такое требование может быть выражено с помощью ограничения с шестью переменными  $AllDiff(F, T, U, W, R, O)$ . Иным образом это требование может быть представлено в виде коллекций бинарных ограничений, таких как  $F \neq T$ . Ограничения сложения для четырех столбцов этой головоломки также включают несколько переменных и могут быть записаны следующим образом:

$$\begin{aligned} O + O &= R + 10 \cdot X_1 \\ X_1 + W + W &= U + 10 \cdot X_2 \\ X_2 + T + T &= O + 10 \cdot X_3 \\ X_3 &= F \end{aligned}$$

где  $X_1$ ,  $X_2$  и  $X_3$  — **вспомогательные переменные**, представляющие цифру (0 или 1), которая переносится в следующий столбец. Ограничения высокого порядка могут быть представлены в виде **гиперграфа ограничений**, подобного приведенному на рис. 5.2, б. Внимательный читатель заметит, что ограничение  $AllDiff$  может быть разбито на бинарные ограничения —  $F \neq T$ ,  $F \neq U$  и т.д. И действительно, в упр. 5.11 предлагается доказать, что каждое ограничение высокого порядка с конечной областью определения можно свести к множеству бинарных ограничений, введя достаточно большое количество вспомогательных переменных. В связи с этим в данной главе будут рассматриваться только бинарные ограничения.



a)

б)

*Рис. 5.2. Пример криптоарифметической задачи: каждая буква обозначает отдельную цифру; задание состоит в том, чтобы найти такую замену букв цифрами, чтобы результирующее выражение суммирования было арифметически правильным, с дополнительным ограничением, что наличие ведущих нулей не допускается (а); гиперграф ограничений для этой криптоарифметической задачи, на котором показано ограничение Alldiff, а также ограничения сложения по столбцам; каждое ограничение обозначено квадратом, который соединен с ограничивающими им переменными (б)*

Все ограничения, рассматривавшиеся до сих пор, были **абсолютными** ограничениями, нарушение которых равносильно тому, что какое-то потенциальное решение больше не рассматривается как такое. С другой стороны, во многих реальных задачах CSP применяются ограничения **предпочтения**, которые указывают, какие решения являются предпочтительными. Например, в задаче составления расписания занятий в университете может потребоваться учесть, что профессор  $X$  предпочитает проводить занятия по утрам, а профессор  $Y$  — после полудня. Расписание занятий, в котором предусмотрено проведение профессором  $X$  занятия в 14:00, все еще может рассматриваться как решение (если не окажется, что профессор  $X$  должен в это время председательствовать на заседании кафедры), но уже не будет оптимальным. Ограничения предпочтения часто можно закодировать как стоимости присваиваний отдельных переменных; например, за назначение профессору  $X$  послеполуденного интервала придется заплатить 2 пункта, которые будут учтены в суммарном значении целевой функции, в то время как утренний интервал имеет стоимость 1 пункта. При использовании такой формулировки задачи CSP с предпочтениями можно решать, используя методы поиска с оптимизацией, либо основанные на поиске пути, либо локальные. Подобные задачи CSP в данной главе больше не рассматриваются, но в разделе с библиографическими заметками приведены некоторые ссылки.

## 5.2. ПРИМЕНЕНИЕ ПОИСКА С ВОЗВРАТАМИ ДЛЯ РЕШЕНИЯ ЗАДАЧ CSP

В предыдущем разделе приведена формулировка задач CSP в виде задач поиска. Если используется такая формулировка, то появляется возможность решать задачи

CSP с помощью любых алгоритмов поиска, приведенных в главах 3 и 4. Предположим, что мы применяем алгоритм поиска в ширину к универсальной формулировке задачи CSP, приведенной в предыдущем разделе. Но мы быстро обнаружим нечто ужасное: коэффициент ветвления на верхнем уровне равен  $nd$ , поскольку любое из  $d$  значений может быть присвоено любой из  $n$  переменных. На следующем уровне коэффициент ветвления равен  $(n-1)d$  и т.д., на всех  $n$  уровнях. При этом создается дерево с  $n! \cdot d^n$  ветвями, даже несмотря на то, что имеется только  $d^n$  возможных полных присваиваний!

В применяемой нами формулировке задачи, которая внешне казалась разумной, но оказалась плохо продуманной, не было учтено существенно важное свойство, общее для всех задач CSP, — **коммутативность**. Задача называется **коммутативной**, если порядок применения любого конкретного множества действий в процессе ее решения не влияет на результат. Именно это свойство характерно для задач CSP, поскольку, присваивая значения переменным, мы достигаем одного и того же частичного присваивания независимо от порядка присваивания. Поэтому *во всех алгоритмах поиска CSP преемники формируются с учетом возможных присваиваний только для единственной переменной в каждом узле дерева поиска*. Например, в корневом узле дерева поиска в задаче раскрашивания карты Австралии можно иметь выбор между  $SA=red$ ,  $SA=green$  и  $SA=blue$ , но мы никогда не должны выбирать между  $SA=red$  и  $WA=blue$ . Определив такое условие, можно надеяться сократить количество листьев до  $d^n$ .

Поиск в глубину, в котором каждый раз выбираются значения для одной переменной и выполняется возврат, если больше не остается допустимых значений, которые можно было бы присвоить переменной, называется **поиском с возвратами**. Этот алгоритм поиска приведен в листинге 5.1. Обратите внимание на то, что в этом алгоритме по существу используется метод инкрементного формирования преемников по одному преемнику за один раз, который описан на с. 131. Кроме того, для формирования преемника текущее присваивание дополняется, а не копируется. Поскольку это представление задач CSP стандартизировано, то в функции Backtracking-Search не требуется учитывать начальное состояние, функцию определения преемника или проверку цели, характерные для рассматриваемой проблемной области. Часть дерева поиска для задачи раскраски карты Австралии показана на рис. 5.3, где значения присваиваются переменным в порядке  $WA, NT, Q, \dots$ .

**Листинг 5.1.** Простой алгоритм поиска с возвратами для решения задач удовлетворения ограничений *csp*. Этот алгоритм составлен по принципу рекурсивного поиска в глубину, описанного в главе 3. Функции **Select-Unassigned-Variable** и **Order-Domain-Values** могут использоваться для реализации эвристических функций общего назначения, которые рассматриваются в тексте данной главы

---

```

function Backtracking-Search(csp) returns решение result
  или индикатор отказа failure
  return Recursive-Backtracking({}, csp)

function Recursive-Backtracking(assignment, csp) returns решение result
  или индикатор отказа failure
  if присваивание assignment является полным then return assignment
  var  $\leftarrow$  Select-Unassigned-Variable(Variables[csp], assignment, csp)

```

```

for each value in Order-Domain-Values(var, assignment, csp) do
    if значение value является совместимым с присваиванием
        assignment согласно ограничениям Constraints[csp] then
            добавить {var = value} к присваиванию assignment
            result  $\leftarrow$  Recursive-Backtracking(assignment, csp)
            if result  $\neq$  failure then return result
            удалить {var = value} из присваивания assignment
return failure

```

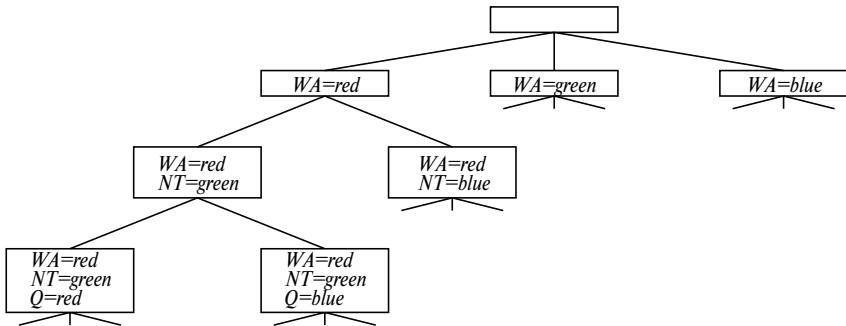


Рис. 5.3. Часть дерева поиска, сформированного путем простого поиска с возвратами при решении задачи раскрашивания карты, приведенной на рис. 5.1

Согласно определениям, приведенным в главе 3, алгоритм простого поиска с возвратами представляет собой неинформированный алгоритм, поэтому не следует рассчитывать на то, что он окажется очень эффективным при решении крупных задач. Результаты его применения к некоторым примерам задач показаны в первом столбце табл. 5.1 и подтверждают эти предположения.

В главе 4 было показано, что такой недостаток неинформированных алгоритмов поиска, как низкая производительность, можно устраниТЬ, предусмотрев использование в них эвристических функций, соответствующих данной проблемной области, которые основаны на наших знаниях о данной задаче. Как оказалось, задачи CSP можно решать эффективно без таких знаний о конкретной проблемной области. Вместо этого в данной главе будут разработаны методы общего назначения, позволяющие найти ответ на перечисленные ниже вопросы.

- Какой переменной должно быть присвоено значение в следующую очередь и в каком порядке необходимо пытаться присваивать эти значения?
- Как влияют текущие присваивания значений переменным на другие переменные с неприсвоенными значениями?
- Если какой-то путь оказался неудачным (т.е. достигнуто состояние, в котором ни одна переменная не имеет допустимых значений), позволяет ли поиск избежать повторения этой неудачи при прохождении последующих путей?

В приведенных ниже подразделах по очереди даны ответы на каждый из этих вопросов.

**Таблица 5.1.** Результаты применения различных алгоритмов CSP для решения разных задач. Слева направо показаны алгоритмы простого поиска с возвратами, поиска с возвратами на основе эвристической функции MRV, локального поиска с предварительной проверкой, локального поиска с предварительной проверкой на основе MRV и локального поиска с минимальными конфликтами. В каждой ячейке показано среднее количество проверок совместимости (за пять прогонов), которые потребовались для решения данной задачи; обратите внимание на то, что все эти числа, кроме двух, находящихся вверху справа, выражены в тысячах (к). Числа в круглых скобках показывают, что за назначенное количество проверок не было найдено ни одного ответа. Первой задачей является поиск раскраски в 4 цвета для 50 штатов США. Остальные задачи взяты из [56, табл. 1]. Во второй задаче подсчитывается общее количество проверок, необходимых для решения всех задач с  $n$  ферзями для  $n$  от 2 до 50. Третьей задачей является “головоломка с зеброй”, которая описана в упр. 5.13. Последние две задачи представляют собой искусственные задачи, формируемые случайным образом (алгоритм с минимальными конфликтами к ним не применялся). Эти результаты показывают, что предварительная проверка на основе эвристической функции MRV является лучшим способом решения всех этих задач по сравнению с другими алгоритмами поиска с возвратами, но этот метод не всегда превосходит локальный поиск с минимальными конфликтами

Задача	Поиск с возвратами	Поиск с возвратами на основе MRV	Локальный поиск с предварительной проверкой	Локальный поиск с предварительной проверкой на основе MRV	Локальный поиск с минимальными конфликтами
Определение раскраски в 4 цвета для 50 штатов США	(> 1000K)	(> 1000K)	2K	60	64
Задача с $n$ ферзями	(> 40 000K)	13 500K	(> 40 000K)	817K	4K
Головоломка с зеброй	3859K	1K	35K	0,5K	2K
Сформированная случайным образом задача 1	415K	3K	26K	2K	–
Сформированная случайным образом задача 2	942K	27K	77K	15K	–

### Упорядочение переменных и значений

Приведенный выше алгоритм поиска с возвратами содержит следующую строку:

```
var ← Select-Unassigned-Variable(Variables[csp], assignment, csp)
```

По умолчанию функция `Select-Unassigned-Variable` выбирает следующую переменную с неприсвоенным значением в порядке, указанном в списке `Variables[csp]`. Такое статическое упорядочение переменных редко приводит к наиболее эффективному поиску. Например, после присваиваний  $WA=red$  и  $NT=green$  остается только одно возможное значение для  $SA$ , поэтому имеет смысл на следующем этапе выполнить присваивание  $SA=blue$ , а не присваивать значение переменной  $Q$ . В действительности после присваивания значения переменной  $SA$  все варианты выбора значений для  $Q$ ,  $NSW$  и  $V$  становятся вынужденными. Эта интуитивная идея (согласно которой в первую очередь следует выбирать переменную с наименьшим количеством “допустимых” значений) называется эвристикой с **минимальным количеством оставшихся значений** (Minimum Remaining Values —

MRV). Эту эвристику называют также эвристикой с “переменной, на которую распространяется наибольшее количество ограниченной”, или эвристикой “до первого неудачного завершения”; последнее название применяется потому, что такая эвристическая функция предусматривает выбор переменной, которая с наибольшей вероятностью вскоре приведет к неудаче, усекая тем самым дерево поиска. Если существует переменная  $X$  с нулевым количеством оставшихся допустимых значений, эвристическая функция MRV выберет  $X$  и неудача будет обнаружена немедленно; это позволяет избежать бессмысленных поисков среди других переменных, которые всегда будут оканчиваться неудачей после того, как в конечном итоге будет выбрана переменная  $X$ . Производительность поиска с использованием этой эвристической функции показана во втором столбце табл. 5.1, обозначенном как *Поиск с возвратами на основе MRV*. Производительность поиска повышается от 3 до 3000 раз по сравнению с простым поиском с возвратами, в зависимости от задачи. Следует отметить, что в применяемом здесь показателе производительности не учитывается дополнительная стоимость вычисления значений эвристической функции; в следующем подразделе описан метод, который позволяет удерживать это значение стоимости в приемлемых рамках.

Эта эвристическая функция MRV вообще не оказывает никакой помощи при выборе для окрашивания в Австралии первого региона, поскольку первоначально каждый регион имеет три допустимых цвета. В таком случае удобной становится **степенная эвристика**. В этой эвристике предпринимается попытка уменьшить коэффициент ветвления в будущих вариантах за счет выбора переменной, которая участвует в наибольшем количестве ограничений на другие переменные с неприсвоенными значениями. На рис. 5.1 переменной с наибольшей степенью, 5, является переменная  $SA$ ; другие переменные имеют степень 2 или 3, за исключением  $T$ , которая имеет степень 0. В действительности после выбора переменной  $SA$  применение степенной эвристики позволяет решить задачу без каких-либо неудачных этапов — теперь можно выбрать любой согласованный цвет в каждой точке выбора и все равно прийти к решению без поиска с возвратами. Эвристика с минимальным количеством оставшихся значений обычно обеспечивает более мощное руководство, но степенная эвристика может оказаться полезной в качестве средства разрешения неопределенных ситуаций.

После выбора одной из переменных в этом алгоритме необходимо принять решение о том, в каком порядке должны проверяться ее значения. Для этого в некоторых случаях может оказаться эффективной эвристика с **наименее ограничительным значением**. В ней предпочтается значение, в котором из рассмотрения исключается наименьшее количество вариантов выбора значений для соседних переменных в графе ограничений. Например, предположим, что на рис. 5.1 сформировано частичное присваивание с  $WA=red$  и  $NT=green$  и что следующий вариант выбора предназначен для  $Q$ . Синий цвет был бы плохим вариантом, поскольку исключил бы последнее допустимое значение, оставшееся для соседней переменной относительно  $Q$ , т.е. переменной  $SA$ . Поэтому эвристика с “наименее ограничительным значением” предпочитает значение  $red$ , а не  $blue$ . Вообще говоря, в этой эвристике предпринимается попытка сохранить максимальную гибкость для последующих присваиваний значений переменным. Безусловно, если бы мы стремились найти все решения некоторой задачи, а не первое из них, то такое упорядочение не имело бы значения,

поскольку нам так или иначе пришлось бы рассмотреть каждое значение. Такое же замечание остается справедливым, если данная задача вообще не имеет решений.

## Распространение информации с помощью ограничений

До сих пор в рассматриваемом алгоритме поиска ограничения, распространяющиеся на какую-то переменную, учитывались только в тот момент, когда происходил выбор этой переменной с помощью функции `Select-Unassigned-Variable`. Но рассматривая некоторые ограничения на предыдущих этапах поиска или даже до начала поиска, можно резко уменьшить пространство поиска.

### Предварительная проверка

Один из способов лучшего использования ограничений во время поиска получил название **предварительной проверки** (forward checking). При каждом присваивании значения переменной  $X$  в процессе предварительной проверки просматривается каждая переменная  $Y$  с неприсвоенным значением, которая соединена с  $X$  с помощью некоторого ограничения, и из области определения переменной  $Y$  удаляется любое значение, которое является несовместимым со значением, выбранным для  $X$ . На рис. 5.4 показан ход поиска решения задачи раскрашивания карты с помощью предварительной проверки. На основании данного примера можно сделать два важных вывода. Прежде всего следует отметить, что после присваивания  $WA=red$  и  $Q=green$  области определения переменных  $NT$  и  $SA$  сокращаются до единственного значения; таким образом, ветвление, связанное с поиском значений для этих переменных, было полностью устранено путем распространения информации, касающейся переменных  $WA$  и  $Q$ . Применение эвристике MRV, которая, безусловно, хорошо сочетается с предварительной проверкой, позволяет на следующем этапе автоматически выбрать значение для переменных  $SA$  и  $NT$ . (Разумеется, что предварительная проверка может рассматриваться как эффективный способ инкрементного вычисления той информации, которая требуется эвристике MRV для выполнения своей работы.) Второй вывод, заслуживающий внимания, состоит в том, что после присваивания  $V=blue$  область определения  $SA$  становится пустой. Поэтому предварительная проверка позволила обнаружить, что частичное присваивание  $\{WA=red, Q=green, V=blue\}$  является несовместимым с ограничениями этой задачи, значит, алгоритм немедленно выполняет возврат.

	$WA$	$NT$	$Q$	$NSW$	$V$	$SA$	$T$
	R G B	R G B	R G B	R G B	R G B	R G B	R G B
Начальные области определения	(R)	G B	R G B	R G B	R G B	G B	R G B
После присваивания $WA=red$	(R)	B	(G)	R B	R G B	B	R G B
После присваивания $Q=green$	(R)	B	(G)	R		(B)	R G B
После присваивания $V=blue$	(R)	B	(G)	R			R G B

Рис. 5.4. Поиск решения задачи с раскрашиванием карты на основе предварительной проверки. Вначале выполняется присваивание  $WA=red$ , затем предварительная проверка приводит к удалению значений  $red$  из областей определения соседних переменных  $NT$  и  $SA$ . После присваивания  $Q=green$  значение  $green$  удаляется из областей определения  $NT$ ,  $SA$  и  $NSW$ . После присваивания  $V=blue$  из областей определения  $NSW$  и  $SA$  удаляется значение  $blue$ , в результате чего переменная  $SA$  остается без допустимых значений

### Распространение ограничения

Хотя предварительная проверка обнаруживает много несовместимостей, она не позволяет обнаружить их все. Например, рассмотрим третью строку на рис. 5.4, которая показывает, что если переменная *WA* имеет значение *red*, а *Q* — *green*, то обеим переменным, *NT* и *SA*, должно быть присвоено значение *blue*. Но соответствующие им регионы являются смежными и поэтому не могут иметь одно и то же значение цвета. Предварительная проверка не позволяет обнаружить эту ситуацию как несовместимость, поскольку не предусматривает достаточно далекий просмотр наперед.  **Распространение ограничения** (constraint propagation) — это общее название методов распространения на другие переменные последствий применения некоторого ограничения к одной переменной; в данном случае необходимо распространить ограничение с *WA* и *Q* на *NT* и *SA* (как было сделано с помощью предварительной проверки), а затем — на ограничение между *NT* и *SA*, чтобы обнаружить указанную несовместимость. К тому же желательно, чтобы такая операция выполнялась быстро: нет смысла ограничивать таким образом объем поиска, если будет расходоваться больше времени на распространение ограничений, чем на выполнение простого поиска.

Идея проверки  **совместимости дуг** легла в основу быстрого метода распространения ограничений, который является значительно более мощным по сравнению с предварительной проверкой. В данном случае термин *дуга* обозначает ориентированное ребро в графе ограничений, такое как дуга от *SA* до *NSW*. Если рассматриваются текущие области определения *SA* и *NSW*, то дуга является совместимой, если для каждого значения *x* переменной *SA* существует некоторое значение *y* переменной *NSW*, которое совместимо с *x*. В третьей строке на рис. 5.4 текущими областями определения *SA* и *NSW* являются *{blue}* и *{red, blue}* соответственно. При *SA=blue* существует совместимое присваивание для *NSW*, а именно *NSW=red*, поэтому дуга от *SA* до *NSW* совместима. С другой стороны, обратная дуга от *NSW* до *SA* несовместима, поскольку применительно к присваиванию *NSW=blue* не существует совместимого присваивания для *SA*. Этую дугу можно сделать совместимой, удалив значение *blue* из области определения *NSW*.

На том же этапе в процессе поиска проверку совместимости дуг можно также применить к дуге от *SA* до *NT*. Третья строка таблицы, приведенной на рис. 5.4, показывает, что обе переменные имеют область определения *{blue}*. Результатом становится то, что значение *blue* должно быть удалено из области определения *SA*, после чего эта область определения остается пустой. Таким образом, применение проверки совместимости дуг привело к раннему обнаружению той несовместимости, которая не была обнаружена с помощью предварительной проверки, применяемой в чистом виде.

Проверку совместимости дуг можно использовать либо в качестве этапа предварительной обработки перед началом процесса поиска, либо в качестве этапа распространения ограничения (аналогично предварительной проверке) после каждого присваивания во время поиска. (Последний алгоритм иногда называют MAC, сокращенно обозначая метод поддержки совместимости дуг — Maintaining Arc Consistency.) И в том и в другом случае процесс проверки необходимо выполнять повторно, до тех пор, пока не перестанут обнаруживаться какие-либо несовместимости. Это связано с тем, что при удалении (в целях устранения некоторой несовместимости дуг) любого значения из области определения некоторой переменной может

появиться новая несовместимость дуг в тех дугах, которые указывают на эту переменную. В полном алгоритме проверки совместимости дуг, АС-3, используется очередь для отслеживания дуг, которые должны быть проверены на несовместимость (листинг 5.2). Каждая дуга  $(X_i, X_j)$  по очереди “снимается с повестки дня” и проверяется; если из области определения  $X_i$  необходимо удалить какие-либо значения, то каждая дуга  $(X_k, X_i)$ , указывающая на  $X_i$ , должна быть повторно вставлена в очередь для проверки. Сложность проверки совместимости дуг можно проанализировать следующим образом: любая бинарная задача CSP имеет самое большое  $O(n^2)$  дуг; каждая дуга  $(X_k, X_i)$  может быть “внесена в повестку дня” только  $d$  раз, поскольку область определения  $X_i$  имеет самое большое  $d$  значений, доступных для удаления; проверка совместимости любой дуги может быть выполнена за время  $O(d^2)$ , поэтому в наихудшем случае затраты времени составляют  $O(n^2d^3)$ . Хотя такой метод является значительно более дорогостоящим по сравнению с предварительной проверкой, все эти дополнительные затраты обычно окупаются<sup>1</sup>.

**Листинг 5.2. Алгоритм проверки совместимости дуг АС-3.** После применения алгоритма АС-3 либо каждая дуга является совместимой, либо некоторая переменная имеет пустую область определения, указывая на то, что эту задачу CSP невозможно сделать совместимой по дугам (и поэтому данная задача CSP не может быть решена). Обозначение “АС-3” предложено разработчиком данного алгоритма [967], поскольку это — третья версия, представленная в его статье

---

```

function AC-3(csp) returns определение задачи CSP, возможно,
    с сокращенными областями определения переменных
inputs: csp, бинарная задача CSP с переменными  $\{X_1, X_2, \dots, X_n\}$ 
local variables: queue, очередь, состоящая из дуг, которая
    первоначально включает все дуги
    из определения задачи csp

while очередь не пуста do
     $(X_i, X_j) \leftarrow \text{Remove-First}(\text{queue})$ 
    if Remove-Inconsistent-Values( $X_i, X_j$ ) then
        for each  $X_k$  in Neighbors[ $X_i$ ] do
            добавить  $(X_k, X_i)$  к очереди queue

function Remove-Inconsistent-Values( $X_i, X_j$ ) returns значение true тогда
    и только тогда, когда произошло удаление некоторого значения
    removed  $\leftarrow \text{false}$ 
    for each x in Domain[ $X_i$ ] do
        if ни одно из значений y в области определения Domain[ $X_j$ ] не
            позволяет использовать  $(x, y)$  для удовлетворения
            ограничения, установленного между  $X_i$  и  $X_j$ 
            then удалить x из области
                определения Domain[ $X_i$ ]; removed  $\leftarrow \text{true}$ 
    return removed

```

---

Поскольку задачи CSP включают задачу 3SAT в качестве частного случая, не следует рассчитывать на то, что удастся найти алгоритм с полиномиальной временной

---

<sup>1</sup> Алгоритм АС-4, предложенный Мором и Хендерсоном [1068], выполняется за время  $O(n^2d^2)$ ; см. упр. 5.10.

сложностью, позволяющий определить, является ли данная конкретная задача CSP совместимой по дугам. Таким образом, можно сделать вывод, что метод проверки совместимости дуг не позволяет обнаружить все возможные несовместимости. Например, как показано на рис. 5.1, частичное присваивание  $\{WA=red, NSW=red\}$  несовместимо, но алгоритм AC-3 не обнаруживает такую несовместимость. Более строгие формы распространения ограничения можно определить с помощью понятия  **$k$ -совместимости**. Задача CSP является  $k$ -совместимой, если для любого множества  $k-1$  переменных и для любого совместимого присваивания значений этим переменным любой  $k$ -й переменной всегда можно присвоить некоторое совместимое значение. Например, 1-совместимость означает, что совместимой является каждая отдельная переменная сама по себе; это понятие называют также **совместимостью узла**. Далее, 2-совместимость — то же, что и совместимость дуги, а 3-совместимость означает, что любая пара смежных переменных всегда может быть дополнена третьей соседней переменной; это понятие именуется также **совместимостью пути**.

Любой граф называется **строго  $k$ -совместимым**, если он является  $k$ -совместимым, а также  $(k-1)$ -совместимым,  $(k-2)$ -совместимым, … и т.д. вплоть до 1-совместимого. Теперь предположим, что имеется некоторая задача CSP с узлами  $n$ , которая сделана строго  $n$ -совместимой (т.е. строго  $k$ -совместимой для  $k=n$ ). Тогда эту задачу можно решить без возвратов. Для этого вначале можно выбрать совместимое значение для  $X_1$ . В таком случае существует гарантия, что удастся выбрать значение для  $X_2$ , поскольку граф является 2-совместимым, для  $X_3$ , поскольку он — 3-совместимый, и т.д. Для каждой переменной  $X_i$  необходимо выполнить поиск только среди  $d$  значений в ее области определения, чтобы найти значение, совместимое с  $X_1, \dots, X_{i-1}$ . Это означает, что гарантируется нахождение решения за время  $O(nd)$ . Безусловно, за такую возможность также приходится платить: любой алгоритм обеспечения  $n$ -совместимости в наихудшем случае должен требовать времени, экспоненциально зависящего от  $n$ .

Между методами обеспечения  $n$ -совместимости и совместимости дуг существует целый ряд промежуточных методов, выбор которых осуществляется с учетом того, что для выполнения более строгих проверок совместимости потребуется больше времени, но это позволяет получить больший эффект с точки зрения сокращения коэффициента ветвления и обнаружения несовместимых частичных присваиваний. Существует возможность рассчитать такое наименьшее значение  $k$ , что выполнение алгоритма проверки  $k$ -совместимости будет гарантировать решение данной задачи без возвратов (см. раздел 5.4), но применение данного расчета на практике не всегда оправдано. В действительности определение подходящего уровня проверки совместимости в основном относится к области эмпирических методов.

### Обработка специальных ограничений

В реальных задачах часто встречаются некоторые типы ограничений, которые могут обрабатываться с помощью алгоритмов специального назначения, более эффективных по сравнению с методами общего назначения, которые рассматривались до сих пор. Например, ограничение *Alldiff* указывает, что все участвующие в нем переменные должны иметь разные значения (как в криптоарифметической задаче). Одна из простых форм проверки несовместимости для ограничений *Alldiff* применяется следующим образом: если в данном ограничении участвуют  $m$  переменных

и все они вместе взятые имеют  $n$  возможных разных значений, притом что  $m > n$ , то это ограничение не может быть удовлетворено.

Применение данной проверки приводит к созданию следующего простого алгоритма: вначале удалить из ограничения каждую переменную, имеющую одноэлементную область определения, затем удалить значение этой переменной из областей определения оставшихся переменных. Повторять эту операцию до тех пор, пока имеются переменные с одноэлементными областями определения. Если в какой-то момент времени появится пустая область определения или останется больше переменных, чем областей определения, это будет означать, что обнаружена несовместимость.

Этот метод можно применить для обнаружения несовместимости в частичном присваивании  $\{WA=red, NSW=red\}$  на рис. 5.1. Обратите внимание на то, что переменные  $SA$ ,  $NT$  и  $Q$  фактически связаны ограничением  $AllDiff$ , поскольку каждая пара соответствующих регионов должна быть обозначена различными цветами. После применения алгоритма АС-3 в сочетании с этим частичным присваиванием область определения каждой переменной сокращается до  $\{green, blue\}$ . Это означает, что имеются три переменные и только два цвета, поэтому ограничение  $AllDiff$  нарушается. Таким образом, иногда простая процедура проверки совместимости для ограничения более высокого порядка эффективнее по сравнению с процедурой проверки совместимости дуг, применяемой к эквивалентному множеству бинарных ограничений.

Возможно, более важным ограничением высокого порядка является **ресурсное ограничение** (resource constraint), иногда называемое ограничением “самое большое” (*atmost*). Например, допустим, что  $PA_1, \dots, PA_4$  обозначают количество персонала, назначенного для выполнения каждого из четырех заданий. Ограничение, согласно которому может быть всего назначено не больше 10 членов персонала, записывается как  $atmost(10, PA_1, PA_2, PA_3, PA_4)$ . Несовместимость можно обнаружить, проверяя сумму минимальных значений текущих областей определения; например, если каждая переменная имеет область определения  $\{3, 4, 5, 6\}$ , то ограничение *atmost* не может быть удовлетворено. Кроме того, можно принудительно добиться совместимости, удаляя максимальное значение из любой области определения, если оно не совместимо с минимальными значениями других областей определения. Таким образом, если каждая переменная в данном примере имеет область определения  $\{2, 3, 4, 5, 6\}$ , то из каждой области определения можно удалить значения 5 и 6.

При решении крупных задач проверки ресурсных ограничений с целочисленными значениями (таких как задачи снабжения, в которых предусматривается перемещение тысяч людей в сотнях транспортных средств) обычно не существует возможности представлять область определения каждой переменной в виде большого множества целых чисел и постепенно сокращать это множество с применением методов проверки совместимости. Вместо этого области определения представляются в виде верхнего и нижнего пределов и управляются с помощью метода распространения этих пределов. Например, предположим, что имеются два рейса,  $Flight271$  и  $Flight272$ , в которых самолеты имеют соответственно вместимость 165 и 385 пассажиров. Поэтому начальные области определения для количества пассажиров в каждом рейсе определяются следующим образом:

$$Flight271 \in [0, 165] \text{ и } Flight272 \in [0, 385]$$

Теперь допустим, что имеется дополнительное ограничение, согласно которому в этих двух рейсах необходимо перевести 420 человек:

$\text{Flight271} + \text{Flight272} \in [420, 420]$ .

Распространяя ограничения пределов, можно сократить области определения до таких величин:

$\text{Flight271} \in [35, 165]$  и  $\text{Flight272} \in [255, 385]$

Задача CSP называется совместимой с пределами (bounds-consistent), если для каждой переменной  $X$ , а также одновременно для нижнего и верхнего предельных значений  $X$  существует некоторое значение  $Y$ , которое удовлетворяет заданному ограничению между  $X$  и  $Y$  для каждой переменной  $Y$ . Такого рода **распространение пределов** (bounds propagation) широко используется в практических задачах с ограничениями.

### Интеллектуальный поиск с возвратами: поиск в обратном направлении

В алгоритме Backtracking-Search, приведенном в листинге 5.1, применялось очень простое правило, касающееся того, что делать, если какая-то ветвь поиска оканчивается неудачей: вернуться к предыдущей переменной и попытаться использовать для нее другое значение. Такой метод называется **хронологическим поиском с возвратами**, поскольку повторно посещается пункт, в котором было принято последнее по времени решение. В данном подразделе будет показано, что существуют намного лучшие способы поиска с возвратами.

Рассмотрим, что произойдет в случае применения простого поиска с возвратами в задаче, показанной на рис. 5.1, с постоянным упорядочением переменных  $Q, NSW, V, T, SA, WA, NT$ . Предположим, что сформировано частичное присваивание  $\{Q=\text{red}, NSW=\text{green}, V=\text{blue}, T=\text{red}\}$ . А после попытки присвоить значение следующей переменной,  $SA$ , будет обнаружено, что любое значение нарушает какое-то ограничение. Алгоритм возвращается к узлу  $T$  и пытается назначить новый цвет для Тасмании! Очевидно, что это — бессмысленное действие, поскольку смена цвета Тасмании не позволяет решить проблему с Южной Австралией.

Более интеллектуальный подход к поиску с возвратами состоит в том, чтобы вернуться к одному из множеств переменных, которые стали причиной неудачи. Это множество называется **конфликтным множеством**; в данном случае конфликтным множеством для  $SA$  является  $\{Q, NSW, V\}$ . Вообще говоря, конфликтное множество для переменной  $X$  представляет собой множество переменных с ранее присвоенными значениями, которые связаны с  $X$  ограничениями. Метод **обратного перехода** выполняет обратный переход к переменной с последним по времени присвоенным значением из конфликтного множества; в данном случае в обратном переходе следует перескочить через узел Тасмании и попытаться применить новое значение для  $V$ . Такая операция может быть легко реализована путем модификации алгоритма Backtracking-Search таким образом, чтобы он накапливал данные о конфликтном множестве, одновременно проверяя одно из значений, допустимых для присваивания. Если не будет найдено ни одного допустимого значения, алгоритм должен возвратиться к последнему по времени элементу конфликтного множества и наряду с этим установить индикатор неудачи.

Внимательный читатель уже должен был заметить, что предварительная проверка позволяет определить конфликтное множество без дополнительной работы, поскольку каждый раз, когда процедура предварительной проверки, основанная на присваивании значения переменной  $X$ , удаляет некоторое значение из области оп-

ределения  $Y$ , она должна добавить  $X$  к конфликтному множеству  $Y$ . Кроме того, каждый раз, когда это последнее значение удаляется из области определения  $Y$ , переменные из конфликтного множества  $Y$  добавляются к конфликтному множеству  $X$ . В этом случае после перехода к  $Y$  можно сразу же установить, куда должен быть выполнен обратный переход в случае необходимости.

А проницательный читатель уже должен был заметить нечто странное: обратный переход происходит, когда каждое значение в некоторой области определения конфликтует с текущим присваиванием, но предварительная проверка обнаруживает этот случай и вообще исключает возможность достижения такого узла! В действительности можно показать, что *каждая ветвь, отсекаемая с помощью обратного перехода, отсекается также с помощью предварительной проверки*. Поэтому простой поиск с обратным переходом в сочетании с поиском с предварительной проверкой становится избыточным, а фактически обратный переход избыточен в любом поиске, в котором используется более строгая проверка совместимости, такая как МАС.

Несмотря на замечания, сделанные в предыдущем абзаце, идея, лежащая в основе обратного перехода, остается перспективной, если возврат осуществляется с учетом причин неудачи. При обратном переходе неудача обнаруживается после того, как область определения некоторой переменной становится пустой, но во многих случаях какая-то ветвь поиска становится бесперспективной задолго до того, как это происходит. Еще раз рассмотрим частичное присваивание  $\{WA=\text{red}, NSW=\text{red}\}$  (которое согласно приведенному выше описанию является несовместимым). Предположим, что на следующем этапе предпринимается попытка присваивания  $T=\text{red}$ , а затем осуществляется присваивание значений переменным  $NT, Q, V, SA$ . Известно, что ни одно присваивание не применимо для этих последних четырех переменных, поэтому в конечном итоге не остается доступных значений, которые можно было бы попытаться присвоить переменной  $NT$ . Теперь возникает вопрос, куда выполнить возврат? Обратный переход не может применяться, поскольку в области определения переменной  $NT$  имеются значения, совместимые со значениями, ранее присвоенными переменным, — переменная  $NT$  не имеет полного конфликтного множества из переменных с ранее присвоенными значениями, которое вызвало бы неудачу при попытке присваивания ей значения. Однако известно, что неудачу вызывают четыре переменные,  $NT, Q, V$  и  $SA$ , вместе взятые, поскольку во множестве переменных с ранее присвоенными значениями должны существовать такие переменные, которые непосредственно конфликтуют с этими четырьмя. Эти рассуждения приводят к созданию более глубокого определения понятия конфликтного множества для такой переменной, как  $NT$ : *конфликтным множеством* называется множество переменных с ранее присвоенными значениями, которые, наряду со всеми переменными со значениями, присваиваемыми в дальнейшем, становятся причиной того, что для  $NT$  не существует совместимого решения. В таком случае конфликтное множество состоит из переменных  $WA$  и  $NSW$ , поэтому алгоритм должен выполнить возврат к  $NSW$  и пропустить переменную, соответствующую Тасмании. Алгоритм обратного перехода, в котором используются конфликтные множества, определенный таким образом, называется *алгоритмом обратного перехода, управляемого конфликтами* (*conflict-directed backjumping*).

Теперь необходимо объяснить, как вычисляются эти новые конфликтные множества. Применяемый для этого метод фактически очень прост. “Окончательная” неудача в какой-то ветви поиска всегда возникает из-за того, что область определения некоторой переменной становится пустой; эта переменная имеет типичное

конфликтное множество. В данном примере неудача возникает при присваивании значения переменной  $SA$ , а ее конфликтным множеством является (скажем)  $\{WA, NT, Q\}$ . Выполняется обратный переход к переменной  $Q$ , и эта переменная поглощает конфликтное множество, полученное от  $SA$  (безусловно, после исключения из него самой переменной  $Q$ ), объединяя его со своим собственным прямым конфликтным множеством, представляющим собой  $\{NT, NSW\}$ ; новым конфликтным множеством становится  $\{WA, NT, NSW\}$ . Это означает, что, продолжая поиск от  $Q$ , нельзя найти решение при наличии ранее выполненного присваивания переменным  $\{WA, NT, NSW\}$ . Поэтому обратный переход выполняется к переменной  $NT$ , присваивание значения которой выполнено последним по времени по сравнению с другими этими переменными. Переменная  $NT$  поглощает множество  $\{WA, NT, NSW\} - \{NT\}$ , объединяя его со своим собственным прямым конфликтным множеством  $\{WA\}$ , что приводит к получению  $\{WA, NSW\}$  (как было указано в предыдущем абзаце). Теперь алгоритм осуществляет обратный переход к  $NSW$ , как и предполагалось. Подведем итог: допустим, что  $X_j$  — текущая переменная, а  $conf(X_j)$  — ее конфликтное множество. Если все попытки присваивания каждого возможного значения переменной  $X_j$  оканчиваются неудачей, то необходимо выполнить возврат к переменной  $X_i$  с последним по времени присвоенным значением во множестве  $conf(X_j)$  и установить:

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_i\}$$

Обратный переход, управляемый конфликтами, позволяет вернуться в правильную точку дерева поиска, но не предотвращает возможности возникновения таких же ошибок в другой ветви того же дерева. Метод **определения ограничений с помощью обучения** (constraint learning) по сути представляет собой метод модификации задачи CSP путем добавления нового ограничения, выдвинутого на основе логического анализа этих конфликтов.

### 5.3. ПРИМЕНЕНИЕ ЛОКАЛЬНОГО ПОИСКА ДЛЯ РЕШЕНИЯ ЗАДАЧ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ

Как оказалось, алгоритмы локального поиска (см. раздел 4.3) являются очень эффективными средствами решения многих задач CSP. В них используется следующая формулировка с полным состоянием: в начальном состоянии присваивается значение каждой переменной, а функция определения преемника обычно действует по принципу изменения за один раз значения одной переменной. Например, в задаче с восемью ферзями начальное состояние может представлять собой случайную конфигурацию из 8 ферзей, стоящих на 8 столбцах, а функция определения преемника выбирает одного ферзя и рассматривает возможность перемещения его на какую-то другую клетку в том же столбце. Еще одна возможность предусматривает, чтобы поиск начался с 8 ферзей (по одному на каждом столбце) в виде какой-то перестановки из 8 строк, а преемник формировался путем обмена двумя строками с двумя ферзями<sup>2</sup>. В этой книге уже фактически был приведен пример применения

<sup>2</sup> Локальный поиск можно легко дополнить для использования его в задачах CSP с помощью целевой функции. В таком случае для оптимизации целевой функции могут применяться все методы поиска с восхождением к вершине и с эмуляцией отжига.

локального поиска для решения задачи CSP — применение поиска с восхождением к вершине для решения задачи с  $n$  ферзями (с. 177). Еще одним примером может служить использование алгоритма WalkSAT (с. 319) для решения задач удовлетворения ограничений, которые представляют собой частный случай задач CSP.

Наиболее очевидная эвристика, которая может применяться при выборе нового значения для какой-либо переменной, состоит в определении того, приводят ли выбранное значение к минимальному количеству конфликтов с другими переменными; таковой является эвристика с ~~и~~ **минимальными конфликтами**. Соответствующий алгоритм приведен в листинге 5.3, пример его применения в задаче с восемью ферзями схематически показан на рис. 5.5, а полученные при этом результаты даны в табл. 5.1.

**Листинг 5.3. Алгоритм Min-Conflicts**, применяемый для решения задач CSP с помощью локального поиска. Начальное состояние может быть выбрано либо случайным образом, либо с помощью жадного процесса присваивания, в котором выбирается значение с минимальными конфликтами для каждой переменной по очереди. Функция **Conflicts** вычисляет количество ограничений, нарушенных данным конкретным значением, с учетом остальной части текущего присваивания

---

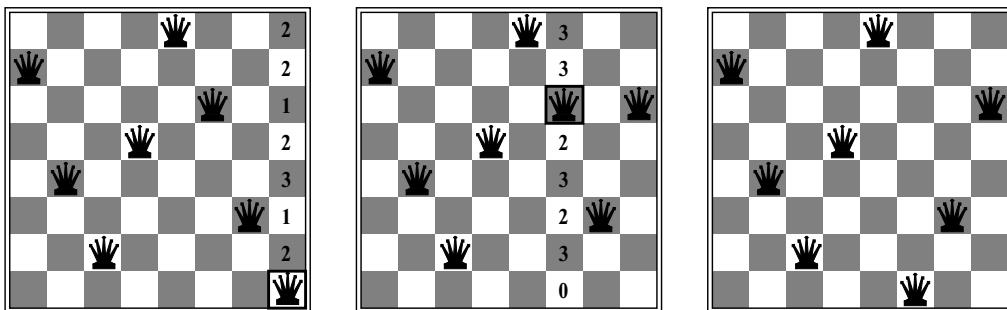
```

function Min-Conflicts(csp, max_steps) returns решение current
    или индикатор неудачи failure
inputs: csp, определение задачи удовлетворения ограничений
        max_steps, количество шагов, которые разрешается
            выполнить,
            прежде чем отказаться от дальнейших попыток

current  $\leftarrow$  первоначальное полное присваивание для csp
for i = 1 to max_steps do
    if current представляет собой решение для задачи csp
        then return current
    var  $\leftarrow$  выбранная случайным образом, конфликтующая переменная
        из Variables[csp]
    value  $\leftarrow$  значение v для var, которое минимизирует
        значение Conflicts(var, v, current, csp)
    задать значение var=value в решении current
return failure

```

---



**Рис. 5.5. Двухэтапный процесс решения задачи с восемью ферзями на основе эвристики с минимальными конфликтами. На каждом этапе выбирается ферзь для повторного назначения ему места в том же столбце. Количество конфликтов (в данном случае количество атакующих ферзей) показано в каждой клетке. Применяемый алгоритм предусматривает перемещение ферзя в клетку с минимальными конфликтами, разрешая неопределенность с помощью случайного выбора**

Алгоритм с минимальными конфликтами показал себя чрезвычайно эффективным при решении многих задач CSP, особенно при наличии подходящего начального состояния. Его производительность показана в последнем столбце табл. 5.1. Больше всего достойно удивления то, что при решении задачи с  $n$  ферзями время прогона алгоритма с минимальными конфликтами остается почти независимым от размера задачи (если не учитываются затраты времени на первоначальную расстановку ферзей). Данный алгоритм решает в среднем за 50 этапов (после начального присваивания) даже задачу с миллионом ферзей. Это замечательное достижение стало стимулом, побудившим к проведению значительной части исследований по локальному поиску в 1990-х годах, а также позволило уточнить различие между легкими и трудными задачами, которое дополнитель но будет рассматриваться в главе 7. Грубо говоря, задача с  $n$  ферзями является легкой для локального поиска, поскольку решения плотно распределены по всему пространству состояний. Кроме того, алгоритм с минимальными конфликтами успешно применяется при решении трудных задач. Например, он использовался для планирования наблюдений на космическом телескопе Хаббл, что позволило сократить затраты времени на планирование наблюдений, намеченных для проведения в течение одной недели, с трех недель (!) примерно до 10 минут.

Еще одним преимуществом локального поиска является то, что он может использоваться для оперативной корректировки в случае изменения условий задачи. Это особенно важно при решении задач планирования. Недельный график работы авиакомпании может включать тысячи рейсов и десятки тысяч персональных назначений, но из-за плохой погоды в одном аэропорту весь этот график может стать невыполнимым. Поэтому желательно иметь возможность откорректировать график с минимальным количеством изменений. Такую задачу легко выполнить с помощью алгоритма локального поиска, начинающего свою работу с текущего графика. Поиск с возвратами, выполняемый с учетом нового множества ограничений, обычно требует гораздо больше времени и может найти менее удачное решение, требующеенести много изменений в текущий график.

## **5.4. СТРУКТУРА ЗАДАЧ**

---

В данном разделе рассматриваются способы, позволяющие использовать для быстрого поиска решений структуру самой задачи, представленную в виде графа ограничений. Большинство описанных здесь подходов являются очень общими и применимыми для решения других задач, кроме CSP, например задач вероятностного формирования рассуждений. В конечном итоге единственный способ, с применением которого можно надеяться справиться с задачами реального мира, состоит в том, чтобы разложить их на множество подзадач. Еще раз обратившись к рис. 5.1, б с целью определить структуру задачи, можно обнаружить еще один факт: Тасмания не связана с материком<sup>3</sup>. Интуитивно ясно, что задачи раскрашивания Тасмании и рас-

---

<sup>3</sup> Педантичный картограф или патриотически настроенный житель Тасмании мог бы возразить, что Тасманию не следует окрашивать в такой же цвет, как и ее ближайшего соседа по материку, чтобы исключить вероятность, что создастся такое впечатление, будто Тасмания является частью этого штата.

крашивания материка представляют собой **независимые подзадачи** — любое решение для материка, скомбинированное с любым решением для Тасмании, приводит к получению решения для всей карты. В том, что эти подзадачи являются независимыми, можно убедиться, рассматривая **связные компоненты** графа ограничений. Каждый компонент соответствует одной подзадаче  $CSP_i$ . Если присваивание  $S_i$  является решением  $CSP_i$ , то  $\bigcup_i S_i$  является решением  $\bigcup_i CSP_i$ . Почему это так важно? Рассмотрим следующее: предположим, что каждая подзадача  $CSP_i$  имеет  $c$  переменных из общего количества  $n$  переменных, где  $c$  — константа. В таком случае должно быть  $n/c$  подзадач, и для решения каждой из них требуется, самое большое, объем работы  $d^c$ . Поэтому общий объем работы измеряется величиной  $O(d^n/c)$ , которая линейно зависит от  $n$ ; без такой декомпозиции общий объем работы измерялся бы величиной  $O(d^n)$ , которая экспоненциально зависит от  $n$ . Приведем более конкретный пример: разделение булевой задачи  $CSP$  с  $n=80$  на четыре подзадачи с  $c=20$  сокращает продолжительность поиска решения в наихудшем случае от такой величины, которая равна сроку существования всей вселенной, до величины меньше одной секунды.

Поэтому полностью независимые подзадачи являются привлекательными, но встречаются редко. В большинстве случаев подзадачи любой задачи  $CSP$  связаны друг с другом. Простейшим случаем является тот, в котором граф ограничений образует **дерево**: любые две переменные связаны не больше чем одним путем. На рис. 5.6, а показан один из примеров в схематическом виде<sup>4</sup>. Ниже будет показано, что **любая задача  $CSP$  с древовидной структурой может быть решена за время, линейно зависящее от количества переменных**. Этот алгоритм имеет перечисленные ниже этапы.

- Выбрать в качестве корня дерева любую переменную и упорядочить переменные от корня до листьев таким образом, чтобы родительский узел каждого узла в дереве предшествовал этому узлу в таком упорядочении (см. рис. 5.6, б). Обозначить эти переменные по порядку как  $X_1, \dots, X_n$ . Теперь каждая переменная, кроме корня, имеет только одну родительскую переменную.

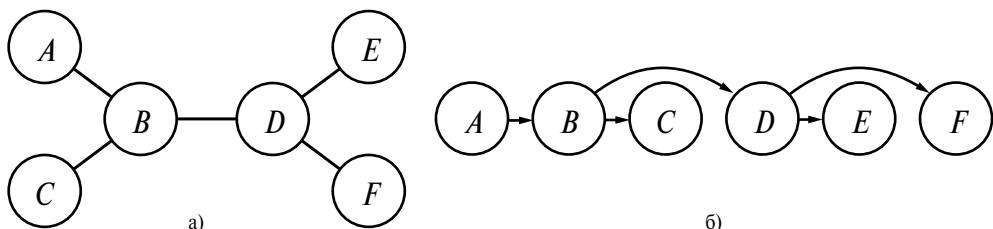


Рис. 5.6. Пример задачи  $CSP$  с древовидной структурой: граф ограничений задачи  $CSP$  с древовидной структурой (а); линейное упорядочение переменных, совместимых с деревом, корнем которого является переменная А (б)

- В цикле по  $j$  от  $n$  до 2 применять проверку совместности к дугам  $(X_i, X_j)$ , где  $X_i$  — родительский узел узла  $X_j$ , удаляя значения из области определения  $Domain[X_i]$  по мере необходимости.

<sup>4</sup> К сожалению, карты с древовидной структурой имеют лишь немногие регионы мира; в качестве одного из возможных исключений следует назвать Сулавеси.

- В цикле по  $j$  от 1 до  $n$  присваивать  $X_j$  любое значение, совместимое со значением, присвоенным  $X_i$ , где  $X_i$  — родительский узел узла  $X_j$ .

Следует учитывать два важных соображения. Во-первых, после выполнения этапа 2 задача CSP становится совместимой по дугам с учетом ориентации дуг, поэтому присваивание значений на этапе 3 не требует возврата. (См. описание понятия  $k$ -совместимости на с. 222.) Во-вторых, благодаря применению проверок совместимости дуг в обратном порядке на этапе 2 алгоритм гарантирует, что никакие удаленные значения не смогут нарушить совместимость тех дуг, которые уже были обработаны. Весь этот алгоритм выполняется за время  $O(nd^2)$ .

Теперь, после создания эффективного алгоритма для деревьев, следует рассмотреть вопрос о том, можно ли каким-то образом приводить к древовидным структурам более общие графы ограничений. Существуют два основных способа выполнения этой задачи; один из них основан на удалении узлов, а другой — на слиянии узлов друг с другом.

Первый подход предусматривает присваивание значений некоторым переменным так, чтобы оставшиеся переменные образовывали дерево. Рассмотрим граф ограничений для карты Австралии, который еще раз показан на рис. 5.7, а. Если с этой карты можно было бы удалить узел  $SA$ , соответствующий Южной Австралии, то граф превратился бы в дерево, как показано на рис. 5.7, б. К счастью, это можно сделать (в графе, а не на самом континенте), зафиксировав некоторое значение для  $SA$  и удалив из областей определения других переменных все значения, несовместимые со значением, выбранным для  $SA$ .

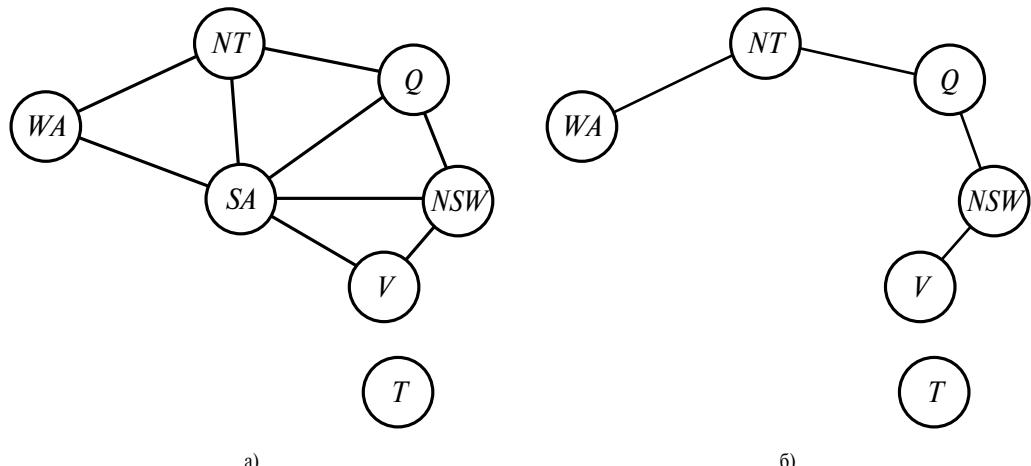


Рис. 5.7. Первый способ преобразования графа ограничений в дерево: первоначальный граф ограничений, впервые приведенный на рис. 5.1 (а); график ограничений после удаления узла  $SA$  (б)

Теперь, после удаления узла  $SA$  и связанных с ним ограничений, любое решение данной задачи CSP будет совместимым со значением, выбранным для  $SA$ . (Такой способ удобен при решении бинарных задач CSP; при наличии ограничений более высокого порядка ситуация усложняется.) Поэтому появляется возможность решить задачу, представленную оставшимся деревом, с помощью приведенного выше алгоритма и таким образом решить всю проблему. Безусловно, в общем случае (в отли-

чие от задачи раскрашивания карты) значение, выбранное для узла  $SA$ , может оказаться неподходящим, поэтому придется проверить каждое из возможных значений. Общий алгоритм решения указанным способом описан ниже.

- Выбрать подмножество  $S$  из множества  $\text{Variables}[csp]$ , такое, что граф ограничений после удаления  $S$  становится деревом. Подмножество  $S$  называется **множеством разрыва цикла** (cycle cutset).
- Для каждого возможного присваивания переменным в  $S$ , которое удовлетворяет всем ограничениям в  $S$ , выполнить следующее:
  - удалить из областей определения оставшихся переменных любые значения, несовместимые с данным присваиванием для  $S$ ;
  - если оставшаяся задача CSP имеет решение, вернуть это решение вместе с присваиванием для  $S$ .

Если множество разрыва цикла имеет размер  $c$ , то общее время прогона алгоритма составляет  $O(\mathcal{A}^c \cdot (n-c) \mathcal{A}^2)$ . В том случае, если граф по своей форме “очень близок к дереву”, то множество  $c$  будет небольшим, а экономия времени по сравнению с прямым поиском с возвратами окажется огромной. Но в наихудшем случае количество элементов  $c$  может достигать  $(n-2)$ . Задача поиска наименьшего множества разрыва цикла является NP-трудной, но известно несколько эффективных алгоритмов решения этой задачи. В целом данный алгоритмический подход носит название **определения условий выбора множества разрыва цикла** (cutset conditioning); мы снова столкнемся с этим понятием в главе 14, где оно используется при формировании рассуждений о вероятностях.

Второй подход основан на формировании **древовидной декомпозиции** (tree decomposition) графа ограничений и создании множества связанных подзадач. Каждая подзадача решается независимо, а затем итоговые решения комбинируются. Как и большинство алгоритмов, действующих по принципу “разделяй и властвуй”, этот алгоритм работает успешно, если подзадачи не слишком велики. На рис. 5.8 показана древовидная декомпозиция задачи раскрашивания карты на пять подзадач. Любая древовидная декомпозиция должна удовлетворять трем приведенным ниже требованиям.

- Каждая переменная из первоначальной задачи появляется по меньшей мере в одной из подзадач.
- Если две переменные первоначальной задачи связаны ограничением, то должны появиться вместе (наряду с этим ограничением) по меньшей мере в одной из подзадач.
- Если какая-то переменная появляется в двух подзадачах в дереве, то должна появляться в каждой подзадаче вдоль пути, соединяющего эти подзадачи.

Первые два условия гарантируют, что в декомпозиции будут представлены все переменные и ограничения. Третье условие на первый взгляд кажется довольно формальным, но просто отражает то ограничение, что любая конкретная переменная должна иметь одно и то же значение в каждой подзадаче, в которой появляется; соблюдение этого ограничения гарантируют связи, соединяющие подзадачи в дереве. Например, переменная  $SA$  появляется во всех четырех связанных подзадачах на рис. 5.8. На основании рис. 5.7 можно убедиться, что эта декомпозиция имеет смысл.

Каждая подзадача решается независимо; если какая-либо из них не имеет решения, то известно, что вся задача также не имеет решения. Если удается решить все подзадачи, то может быть предпринята попытка составить глобальное решение следующим образом. Прежде всего, каждая подзадача рассматривается как “мегапеременная”, областью определения которой является множество всех решений этой подзадачи. Например, самая левая подзадача на рис. 5.8 представляет задачу раскрашивания карты с тремя переменными и поэтому имеет шесть решений; одним из них является  $\{WA=red, SA=blue, NT=green\}$ . Затем решается задача с ограничениями, связывающими подзадачи; для этого используется эффективный алгоритм для деревьев, приведенный выше. Ограничения, связывающие подзадачи, указывают на то, что решения подзадач должны быть согласованными по их общим переменным. Например, если за основу берется решение первой подзадачи  $\{WA=red, SA=blue, NT=green\}$ , то единственным совместимым решением для следующей подзадачи становится  $\{SA=blue, NT=green, Q=red\}$ .

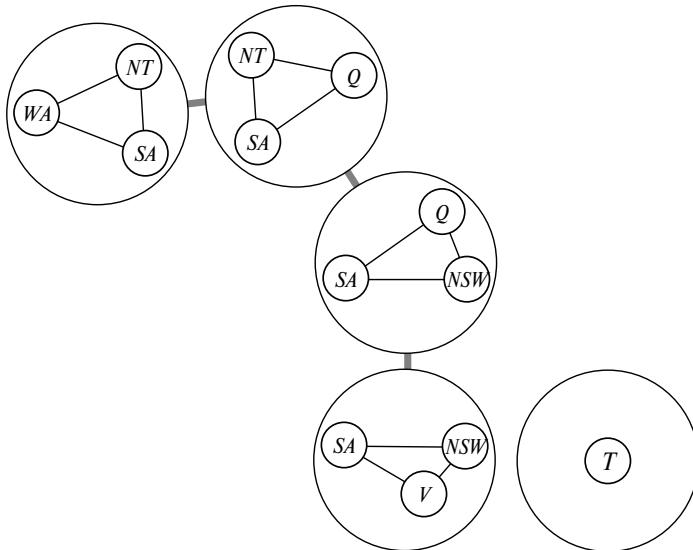


Рис. 5.8. Древовидная декомпозиция графа ограничений, показанного на рис. 5.7, а

Любой конкретный граф ограничений допускает большое количество древовидных декомпозиций; при выборе декомпозиции нужно стремиться к тому, чтобы подзадачи были как можно меньше. **Ширина дерева** древовидной декомпозиции графа на единицу меньше размера наибольшей подзадачи; ширина дерева самого графа определяется как минимальная ширина дерева среди всех его древовидных декомпозиций. Если граф имеет ширину дерева  $w$  и дана соответствующая древовидная декомпозиция, то соответствующая задача может быть решена за время  $O(nd^{w+1})$ . Это означает, что **задачи CSP с графиками ограничений, характеризующимися конечной шириной дерева, могут быть решены за полиномиальное время**. К сожалению, задача поиска декомпозиции с минимальной шириной дерева является NP-трудной, но существуют эвристические методы, которые хорошо работают на практике.

## РЕЗЮМЕ

- **Задачи удовлетворения ограничений** (Constraint Satisfaction Problems — CSP) состоят из переменных с налагаемыми на них ограничениями. В виде задач CSP могут быть описаны многие важные задачи реального мира. Структуру любой задачи CSP можно представить в виде ее **графа ограничений**.
- Для решения задач CSP обычно применяется **поиск с возвратами** — одна из форм поиска в глубину.
- Независимыми от области определения методами выявления того, какая переменная должна быть выбрана следующей в ходе поиска с возвратами, являются эвристики, основанные на определении **минимального количества оставшихся значений**, и **степенные эвристики**. Для упорядочения значений переменной может применяться эвристика с **наименее ограничительным значением**.
- В алгоритме поиска с возвратами можно намного сократить коэффициент ветвления задачи, распространяя последствия частичных присваиваний, сформированных в ходе работы этого алгоритма. Простейшим методом такого распространения является **предварительная проверка**. Более мощным методом является обеспечение **совместности дуг**, но применение этого метода может оказаться более дорогостоящим.
- Возврат происходит после того, как для переменной нельзя больше найти допустимое присваивание. При использовании **обратного перехода, управляемого конфликтами**, возврат происходит непосредственно к источнику проблемы, возникшей в процессе поиска.
- Для решения задач с ограничениями весьма успешно используется локальный поиск на основе эвристики с **минимальными конфликтами**.
- Сложность решения задачи CSP в значительной степени зависит от структуры ее графа ограничений. Задачи с древовидной структурой могут быть решены за линейное время. Метод **определения условий формирования множества разрыва цикла** позволяет преобразовать задачу CSP общего вида в задачу с древовидной структурой и является очень эффективным, если существует возможность найти небольшое множество разрыва цикла. Методы **древовидной декомпозиции** предусматривают преобразование задачи CSP в дерево подзадач и являются эффективными, если **ширина дерева** графа ограничений мала.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

В самых ранних работах, относящихся к проблематике удовлетворения ограничений, главным образом рассматривались числовые ограничения. Выраженные в виде уравнений ограничения с целочисленными областями определения были исследованы индийским математиком Брахмагуптой в VII веке; их часто называют **диофантовыми уравнениями** в честь греческого математика Диофанта (около 200–284 гг.), который в своих исследованиях фактически рассматривал область определения положительных рациональных чисел. Систематические методы решения линейных уравнений путем удаления переменных исследовались Гауссом [527]; истоки

подходов к решению ограничений с линейными неравенствами можно найти в работах Фурье [487].

Задачи удовлетворения ограничений с конечными областями определения также имеют долгую историю. Например, одной из старых задач в математике является **раскрашивание графа** (раскрашивание карты — частный случай данной задачи). Согласно данным, приведенным в [125], гипотезу о четырех цветах (в соответствии с которой любой плоский граф можно раскрасить с помощью четырех или меньшего количества цветов) впервые выдвинул Френсис Гутри, студент де Моргана, в 1852 году. Эта задача не поддавалась решению (несмотря на то, что в некоторых публикациях утверждалось обратное) до тех пор, пока доказательство не было получено с помощью компьютера Аппелем и Хакеном [35].

Специальные классы задач удовлетворения ограничений рассматривались на протяжении всей истории развития компьютерных наук. Одним из самых первых примеров решения таких задач, оказавшим наибольшее влияние, была система Sketchpad [1476], которая решала геометрические ограничения на чертежах и была предшественником современных программ рисования и инstrumentальных средств САПР. Выявлению того, что задачи CSP представляют собой общий класс задач, мы обязаны Уго Монтанари [1073]. Первые попытки приведения задач CSP высокого порядка к чисто бинарным задачам CSP с помощью вспомогательных переменных (см. упр. 5.11) были впервые предприняты в XIX веке логиком Чарльзом Сандерсоном Пирсом. Соответствующие методы были введены в тематику CSP Дектером [367] и доработаны Бакусом и ван Биком [55]. Задачи CSP с предпочтениями между решениями широко рассматривались в исследованиях по оптимизации; обобщение инфраструктуры CSP, позволяющее использовать предпочтения, приведено в [134]. Для решения задач оптимизации может также применяться алгоритм устранения интервалов [369].

Использование поиска с возвратами для удовлетворения ограничений было предложено Битнером и Рейнгольдом [135], но эти ученые проследили истоки идей своего основного алгоритма до работ XIX века. Кроме того, Битнер и Рейнгольд впервые предложили использовать эвристику MRV, названную ими *эвристикой с наиболее ограниченной переменной*. Брелаз [181] использовал степенную эвристику для устранения неопределенности, возникающей после применения эвристики MRV. Полученный в итоге алгоритм, несмотря на его простоту, все еще остается наилучшим методом раскрашивания произвольных графов в  $k$  цветов. Харалик и Эллиот [618] предложили эвристику с наименее ограничительным значением.

Расширению популярности методов распространения ограничений способствовало успешное решение Вальцем [1552] задач полиэдральной линейной разметки для систем компьютерного зрения. Вальц показал, что применение методов распространения ограничений при решении многих задач позволяет полностью исключить необходимость в использовании возвратов. Монтанари [1073] ввел понятие сетей ограничений и предложил метод распространения ограничений на основе понятия совместимости пути. Аллен Макворт [967] предложил алгоритм AC-3 для обеспечения совместимости дуг, а также выдвинул общую идею о том, что поиск с возвратами необходимо сочетать с обеспечением совместимости некоторой степени. Более эффективный алгоритм обеспечения совместимости дуг, AC-4, был разработан Мором и Хендersonом [1068]. Вскоре после появления статьи Макворта исследователи приступили к экспериментам в области поиска компромисса между затратами на обес-

печение совместимости и достигаемыми за счет этого преимуществами с точки зрения сокращения объема поиска. Харалик и Эллиот [618] предпочли минимальный алгоритм предварительной проверки, описанный Макгрегором [1028], а Гашниг [522] предложил применять полную проверку совместимости дуг после присваивания значения каждой переменной; в дальнейшем соответствующему алгоритму в работе Сабина и Фрейдера [1335] было присвоено название MAC. В последней статье представлены некоторые убедительные свидетельства того, что при решении более трудных задач CSP полная проверка совместимости дуг вполне окупается. Фрейдер [497], [498] исследовал понятие  $k$ -совместимости и ее связь со сложностью решения задач CSP. Айт [36] описал универсальную алгоритмическую инфраструктуру, в рамках которой могут быть проанализированы алгоритмы распространения совместимости.

Специальные методы обработки ограничений высокого порядка были созданы в основном в контексте **логического программирования в ограничениях**. Превосходный обзор исследований в этой области приведен в [987]. Ограничение *Alldiff* исследовалось в [1272]. Ограничения с пределами были включены в тематику логического программирования в ограничениях ван Хентенриком и др. [1533].

Основной метод обратного перехода был разработан Джоном Гашнигом [521], [522]. Кондрак и ван Бик [831] показали, что этот алгоритм по сути перекрывается алгоритмом предварительной проверки. Обратный переход, управляемый конфликтами, был предложен Проссером [1240]. Наиболее общая и мощная форма интеллектуального поиска с возвратами фактически разработана еще довольно давно Стальнаном и Зюссманом [1456]. Предложенный ими метод **поиска с возвратами, управляемого зависимостями**, привел к разработке **систем обеспечения истинности** [409], которые будут рассматриваться в разделе 10.8. Связь между этими двумя научными областями проанализирована в [348].

В указанной работе Стальнана и Зюссмана была также предложена идея **регистрации ограничения**, в соответствии с которой частичные результаты, полученные в ходе поиска, можно сохранять и повторно использовать на последующих этапах этого поиска. Данная идея была введена формально в поиск с возвратами Дектером [366]. Особенно простым методом является **проставление обратных отметок** [522], в котором для предотвращения повторной проверки ограничений сохраняются и используются совместимые и несовместимые попарные присваивания. Проставление обратных отметок можно комбинировать с обратным переходом, управляемым конфликтами; в [831] представлен гибридный алгоритм, который пре-восходит любой из этих методов, отдельно взятых (и это подтверждается формальным доказательством). В методе **динамического поиска с возвратами** [558] сохраняются успешные частичные присваивания из полученных позднее подмножеств переменных при поиске с возвратами по предыдущему варианту, который не влияет на дальнейший успех.

Применение локального поиска при решении задач удовлетворения ограничений стало популярным после публикации [799] с описанием метода **эмуляции отжига** (см. главу 4), который широко используется для решения задач планирования. Эвристика с минимальными конфликтами была впервые предложена в [601] и независимо разработана в [1058]. В [1447] показано, как эта эвристика может использоваться для решения задачи с 3 000 000 ферзей меньше чем за минуту. Этот поразительный успех локального поиска на основе эвристики с минимальными конфликтами при решении задачи с  $n$  ферзями привел к переоценке характера и распространения “легких”

и “трудных” задач. В [244] исследовалась сложность задач CSP, сформированных случайным образом, и было обнаружено, что почти все такие задачи являются либо тривиально легкими, либо не имеют решений. “Трудные” экземпляры задач встречаются, только если параметры генератора задач устанавливаются в некотором узком диапазоне, в пределах которого лишь примерно половина задач является разрешимой. Этот феномен рассматривается дополнительно в главе 7.

Исследования, касающиеся структуры и сложности задач CSP, начались с [499], где было показано, что поиск в деревьях с совместимыми дугами происходит без каких-либо возвратов. Аналогичные результаты применительно к ациклическим гиперграфам были получены в области теории баз данных [92]. Со временем публикации этих статей достигнут значительный прогресс в части получения более общих результатов, касающихся связи между сложностью решения задачи CSP и структурой ее графа ограничений. Понятие ширины дерева было введено специалистами по теории графов Робертсоном и Сеймуром [1295]. Дектер и Перл [372], [373], основываясь на работе Фрейдера, применяли то же понятие (названное ими **индуцированной шириной**) к задачам удовлетворения ограничений и разработали подход с древовидной декомпозицией, кратко описанный в разделе 5.4. Опираясь на эту работу и на результаты из области теории баз данных, Готлобб и др. [585], [586] разработали понятие **ширины гипердерева**, которое основано на методе характеризации задачи CSP как гиперграфа. Они не только показали, что любую задачу CSP с шириной гипердерева  $w$  можно решить за время  $O(n^{w+1} \log n)$ , но и обосновали утверждение, что критерий ширины гипердерева превосходит все ранее предложенные критерии “ширины” в том смысле, что в некоторых случаях ширина гипердерева является конечной, тогда как ширина, определяемая другими критериями, — неограниченной.

В литературе можно найти несколько хороших обзоров с описанием методов решения задач CSP, включая [73], [370] и [866], а также энциклопедические сборники статей [368] и [968]. В [1194] приведен обзор разрешимых классов задач CSP и описаны как методы структурной декомпозиции, так и методы, основанные на свойствах областей определения или свойствах самих ограничений. В [831] приведен аналитический обзор алгоритмов поиска с возвратами, а в [56] приведен обзор, характеризующийсяющейся большей практической направленностью. В [987] и [1514] эта тематика рассматривается гораздо более глубоко, чем было возможно ее представить в данной главе. Некоторые интересные приложения описаны в сборнике статей [500], который вышел под редакцией Фрейдера и Маквортса. Статьи на тему удовлетворения ограничений регулярно публикуются в журнале *Artificial Intelligence* и в специализированном журнале *Constraints*. Основным местом встречи специалистов в этой области является конференция *International Conference on Principles and Practice of Constraint Programming*, часто называемая просто *CP*.

## **УПРАЖНЕНИЯ**

---

- 5.1.** Самостоятельно сформулируйте определения следующих понятий: проблема удовлетворения ограничений, ограничение, поиск с возвратами, совместимость дуги, обратный переход и минимизация конфликтов.
- 5.2.** Сколько решений имеет задача раскрашивания карты, показанная на рис. 5.1?

- 5.3. Объясните, почему при поиске решения задачи CSP одна из хороших эвристик состоит в том, что следует выбирать переменную, которая является наиболее ограниченной, и вместе с тем выбирать наименее ограничительное значение.
- 5.4. Рассмотрите задачу составления (а не решения) кроссвордов<sup>5</sup>: подбора слов, которые укладываются в прямоугольную сетку. Эта сетка, которая указывается как часть задания, определяет, какие квадраты пусты и какие затенены. Предположим, что предоставлен список слов (например, словарь) и задача состоит в том, чтобы заполнить пустые квадраты с использованием любого подмножества из этого списка. Точно сформулируйте эту задачу следующими двумя способами.
- Как общую задачу поиска.** Выберите соответствующий алгоритм поиска и определите эвристическую функцию, если, по вашему мнению, она необходима. Как лучше заполнять пустые квадраты: одновременно по одной букве или по одному слову?
  - Как задачу удовлетворения ограничений.** Должны ли переменные быть представлены в виде слов или букв?
- Какая формулировка, по вашему мнению, является наилучшей? Объясните, почему?
- 5.5. Приведите точные формулировки каждой из перечисленных ниже задач в виде задач удовлетворения ограничений.
- Планирование покрытия пола** прямоугольниками. Найти в большом прямоугольнике неперекрывающиеся места для размещения меньших прямоугольников.
  - Составление расписания занятий.** Определены следующие исходные данные: постоянное количество преподавателей и классов, список предлагаемых занятий и список возможных временных интервалов для занятий. С каждым преподавателем связано множество занятий, которые он может проводить.
- 5.6. Решите криптоарифметическую задачу, приведенную на рис. 5.2, вручную, с помощью поиска с возвратами, предварительной проверки, а также на основе эвристик с MRV и наименее ограничительным значением.
- 5.7. В табл. 5.1 приведены результаты проверки производительности различных алгоритмов при решении задачи с  $n$  ферзями. Проведите испытания тех же алгоритмов при решении задач раскрашивания карты, формируемых случайным образом, с помощью следующего метода: разбросать  $n$  точек на единичном квадрате; выбрать точку  $X$  случайным образом, соединить  $X$  прямой линией с ближайшей точкой  $Y$ , такой, что  $X$  еще не соединена с  $Y$ , и соединительная линия не пересекает какую-либо другую линию; повторять предыдущий шаг до тех пор, пока не будет исключена возможность проводить дальнейшие соединения. Сформируйте таблицу производительности для наибольшего значения  $n$ , с которым удастся справиться с помощью этих алгоритмов, используя как  $d=4$ , так и  $d=3$  цветов. Прокомментируйте полученные вами результаты.

<sup>5</sup> В [560] обсуждается несколько методов составления кроссвордов. В [938] рассматривается более сложная задача их решения.

- 5.8.** Воспользуйтесь алгоритмом АС-3, чтобы показать, что проверка совместимости дуг позволяет обнаружить несовместимость частичного присваивания  $\{WA=red, V=blue\}$  для задачи, показанной на рис. 5.1.
- 5.9.** Какова в наихудшем случае времененная сложность при прогоне алгоритма АС-3 применительно к задаче CSP с древовидной структурой?
- 5.10.** Алгоритм АС-3 помещает обратно в очередь каждую дугу  $(X_k, X_i)$  каждый раз, когда любое значение удаляется из области определения  $X_i$ , даже если каждое значение  $X_k$  совместимо с некоторыми оставшимися значениями  $X_i$ . Предположим, что для каждой дуги  $(X_k, X_i)$  отслеживается количество оставшихся значений  $X_i$ , которые являются совместимыми с каждым значением  $X_k$ . Объясните, как можно эффективно обновлять эти числа, и тем самым докажите, что совместимость дуг можно обеспечить за суммарное время  $O(n^2d^2)$ .
- 5.11.** Покажите, как можно преобразовать одно тернарное ограничение, такое как “ $A+B=C$ ”, в три бинарных ограничения с использованием вспомогательной переменной. Может быть принято предположение о конечных областях определения. (*Подсказка.* Предусмотрите использование новой переменной, которая принимает значения, являющиеся парами других значений, и примените такие ограничения, как “ $X$  является первым элементом пары  $Y$ ”.) Затем покажите, как можно трактовать аналогичным образом ограничения больше чем с тремя переменными. Наконец, покажите, как можно устраниТЬ унарные ограничения, модифицируя области определения переменных. На этом завершается демонстрация того, что любую задачу CSP можно преобразовать в задачу CSP только с бинарными ограничениями.
- 5.12.** Предположим, что известно, будто граф имеет множество разрыва цикла, содержащее не больше  $k$  узлов. Опишите простой алгоритм поиска минимального множества разрыва цикла, время прогона которого не намного превышает  $O(n^k)$  для задачи CSP с  $n$  переменными. Найдите в литературе методы обнаружения приближенно минимальных множеств разрыва цикла за время, которое полиномиально зависит от размера множества разрыва. Обеспечивает ли наличие таких алгоритмов практическую применимость метода поиска на основе множества разрыва цикла?
- 5.13.** Рассмотрите приведенную ниже логическую головоломку. В пяти домах, имеющих разный цвет, живут лица пяти национальностей, которые предпочитают разные сорта сигарет, пьют разные напитки и держат разных домашних питомцев. На основе следующих фактов найдите ответ на вопрос: “В каком доме держат зебру и в каком доме пьют воду?”
- Англичанин живет в доме красного цвета.
  - Испанец держит собаку.
  - Норвежец живет в первом доме слева.
  - Сигареты “Кулс” курят в доме желтого цвета.
  - Человек, курящий “Честерфилд”, живет рядом с домом человека, который держит лису.
  - Норвежец живет рядом с домом синего цвета.
  - Человек, курящий “Уинстон”, держит улиток.

- Человек, курящий “Лаки Страйк”, пьет апельсиновый сок.
- Украинец пьет чай.
- Японец курит “Парламент”.
- Сигареты “Кулс” курят в доме, находящемся рядом с домом, где держат лошадь.
- Кофе пьют в доме зеленого цвета.
- Дом зеленого цвета находится непосредственно справа (с точки зрения того, кто решает эту задачу) от дома цвета слоновой кости.
- Молоко пьют в среднем доме.

Обсудите различные представления этой задачи в виде задачи CSP. По каким причинам следовало бы предпочесть одно представление другому?

# 6 ПОИСК В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ

*В этой главе рассматриваются проблемы, возникающие при попытке агента планировать наперед в мире, где другие агенты составляют планы, направленные против него.*

## 6.1. ИГРЫ

В главе 2 представлены **мультиагентные варианты среды**, в которых каждый конкретный агент вынужден принимать во внимание действия других агентов и устанавливать, как они влияют на его собственное благополучие. Непредсказуемость действий этих прочих агентов может потребовать в процессе решения задачи агентом учета многих возможных **непредвиденных ситуаций**, как было описано в главе 3. Различие между **кооперативными** и **конкурентными** мультиагентными вариантами среды также было показано в главе 2. Наличие конкурентных вариантов среды, в которых цели агентов конфликтуют, приводит к возникновению задач **поиска в условиях противодействия**, часто называемых  $\bowtie$  **играми**.

В математической **теории игр**, одной из ветвей экономики, любые мультиагентные варианты среды рассматриваются как игры, при условии, что влияние каждого агента на других является “значительным”, независимо от того, являются ли агенты кооперативными или конкурентными<sup>1</sup>. В искусственном интеллекте “играми” обычно называют довольно специфические формы взаимодействия агентов, которые теоретиками игр именуются как детерминированные, поочередные, охватывающие двух игроков  $\bowtie$  **игры с нулевой суммой и с полной информацией**. В терминологии, принятой в данной книге, это соответствует детерминированным, полностью наблюдаемым вариантам среды, в которых имеются два агента, обязанных чередовать свои действия, и в которых значения полезности в конце игры всегда равны и противоположны. Например, если один игрок выигрывает игру в шахматы (+1), другой игрок обязательно проигрывает (-1). В подобной ситуации условия противо-

---

<sup>1</sup> Варианты среды с очень большим количеством агентов лучше рассматривать как **экономики**, а не как игры.

действия возникают именно из-за такого противопоставления функций полезности агентов. В данной главе будут кратко рассматриваться игры с несколькими игроками, игры с ненулевой суммой и стохастические игры, но надлежащее обсуждение теории игр откладывается до главы 17.

Игры заставляли людей напрягать свои интеллектуальные способности (иногда до угрожающей степени) на протяжении всего существования цивилизации. В силу своего абстрактного характера игры являются привлекательным объектом исследований и в области искусственного интеллекта. Состояние игры можно легко представить, а поведение агентов обычно ограничено небольшим количеством действий, результаты которых определяются с помощью точных правил. Спортивные игры, такие как крокет и хоккей с шайбой, имеют гораздо более сложные описания, значительно больший диапазон возможных действий и довольно неточные правила, определяющие допустимость действий. За исключением проблематики создания робота-футболиста эти спортивные игры не привлекают значительного интереса в обществе специалистов по искусственному интеллекту.

Ведение игр было одной из первых задач, рассматриваемых в области искусственного интеллекта. К 1950 году, почти сразу же после того, как компьютеры стали программируемыми, шахматами уже интересовались Конрад Цузе (изобретатель первого программируемого компьютера и разработчик первого языка программирования), Клод Шенон (основоположник теории информации), Норберт Винер (создатель современной теории управления) и Аллан Тьюринг. С тех пор уровень игры с применением компьютеров неуклонно повышался и достиг того, что компьютеры превзошли людей в шашках и игре “Отелло” (“реверси”), побеждали чемпионов (но не всегда) в шахматах и нардах, а также стали конкурентоспособными во многих других играх. Основным исключением остается игра го, в которой компьютеры пока еще выступают на любительском уровне.

Игры, в отличие от большинства учебных задач, которые рассматривались в главе 3, интересны тем, что в них очень трудно найти решение. Например, шахматы характеризуются в среднем коэффициентом ветвления, примерно равным 35, а игра часто продолжается до 50 ходов со стороны каждого игрока, поэтому дерево поиска имеет приблизительно  $35^{100}$  или  $10^{154}$  узлов (хотя граф поиска включает “всего лишь” около  $10^{40}$  различных узлов). Поэтому игры, как и реальная жизнь, требуют способности принимать хоть какие-то решения, даже если вычисление оптимального решения неосуществимо. Кроме того, игры сурово наказывают за неэффективность. Притом что реализация поиска A\*, в два раза менее эффективная по сравнению с другой реализацией, просто потребует вдвое больше времени для получения окончательного решения, шахматная программа, вдвое менее эффективно использующая отведенное ей время, по-видимому, потерпит поражение на самых ранних этапах игры, даже при всех прочих равных условиях. Поэтому исследователи, работающие в области ведения игр, стали авторами многих интересных идей, касающихся того, как обеспечить наилучшее возможное использование времени.

Начнем описание данной темы с определения понятий оптимального хода игры и алгоритма его поиска. Затем рассмотрим методы выбора хорошего хода в условиях ограниченного времени. **Отсечение** позволяет игнорировать те части дерева поиска, которые не оказывают влияния на окончательный выбор, а эвристические **функции оценки** позволяют приближенно рассчитывать истинную полезность состояния без проведения полного поиска. В разделе 6.5 рассматриваются такие игры, включающие элемент случайности, как нарды; кроме того, в данной главе рассматривается

бридж, который включает элементы **неполной информации**, поскольку не все карты видны каждому игроку. Наконец, в этой главе будет описано, как новейшие программы ведения игр постепенно преодолевают сопротивление людей в борьбе с этими программами и каковы направления будущих разработок.

## **6.2. ПРИНЯТИЕ ОПТИМАЛЬНЫХ РЕШЕНИЙ В ИГРАХ**

---

В данной главе речь идет об играх с двумя игроками, которые будут именоваться MAX и MIN по причинам, которые вскоре станут очевидными. Игрок MAX ходит первым, после чего игроки делают ходы по очереди до тех пор, пока игра не закончится. В конце игры игроку-победителю присваиваются очки, а на побежденного налагается штраф. Игра может быть формально определена как своего рода задача поиска с описанными ниже компонентами.

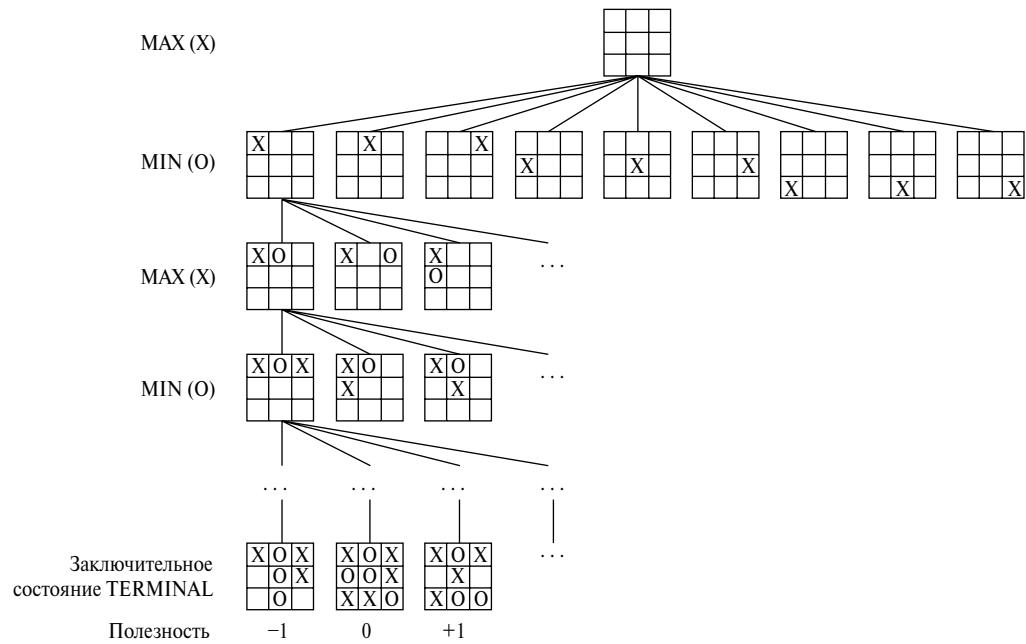
- **Начальное состояние**, которое включает позицию на доске и определяет игрока, который должен ходить.
- **Функция определения преемника**, возвращающая список пар (*move, state*), каждая из которых указывает допустимый ход и результирующее состояние.
- **Проверка терминального состояния**, которая определяет, что игра закончена. Состояния, в которых игра закончена, называются **терминальными состояниями**.
- **Функция полезности** (называемая также *целевой функцией*, или *функцией вознаграждения*), которая сообщает числовое значение терминальных состояний. В шахматах результатом является победа, поражение или ничья, со значениями +1, -1 или 0. Некоторые игры имеют более широкий диапазон возможных результатов; например, количество очков в нардах может составлять от +192 до -192. В настоящей главе в основном рассматриваются игры с нулевой суммой, хотя будут также кратко упоминаться игры с ненулевой суммой.

Начальное состояние и допустимые ходы каждой стороны определяют **дерево игры** для данной игры. На рис. 6.1 показана часть дерева игры в крестики-нолики. Из начального состояния игрок MAX имеет девять возможных ходов. Ходы чередуются так, что MAX ставит значок X, а MIN значок O, до тех пор пока не будут достигнуты листовые узлы, соответствующие терминальным состояниям, таким, что один игрок поставил три своих значка в один ряд или заполнены все клетки. Число под каждым листовым узлом указывает значение полезности соответствующего терминального состояния с точки зрения игрока MAX; предполагается, что высокие значения являются благоприятными для игрока MAX и неблагоприятными для игрока MIN (именно поэтому в теории этим игрокам присвоены такие имена). Задача игрока MAX состоит в использовании дерева поиска (особенно данных о полезности терминальных состояний) для определения наилучшего хода.

### **Оптимальные стратегии**

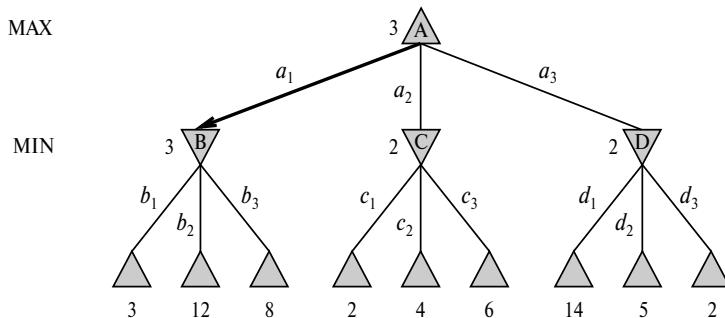
При решении обычных задач поиска оптимальное решение для игрока MAX должно представлять собой последовательность ходов, ведущих к цели — к терминальному состоянию, которое соответствует выигрышу. С другой стороны, в игре

участвует также игрок MIN, который имеет другое мнение по этому поводу. Это означает, что игрок MAX должен найти надежную **стратегию**, позволяющую определить ход игрока MAX в начальном состоянии, затем ходы игрока MAX в состояниях, ставших результатом любого возможного ответа игрока MIN, а затем ходы MAX в состояниях, ставших результатом любого возможного ответа MIN на те ходы, и т.д. Грубо говоря, оптимальная стратегия приводит к итогу, по меньшей мере, такому же благоприятному, как и любая другая стратегия, в тех условиях, когда приходится играть с противником, не допускающим ошибок. Прежде всего рассмотрим, как найти эту оптимальную стратегию, даже притом что для MAX часто будет неосуществимой задача ее исчерпывающего вычисления в играх, более сложных, чем крестики-нолики.



*Рис. 6.1. (Частичное) дерево поиска для игры крестики-нолики. Верхний узел представляет собой начальное состояние, а первым ходит игрок MAX, ставя значок X в пустой клетке. На этом рисунке показана часть дерева поиска, в которой демонстрируются чередующиеся ходы игроков MIN (значок O) и MAX (значок X). Ходы выполняются до тех пор, пока в конечном итоге не будет достигнуто одно из терминальных состояний, которым могут быть назначены данные о полезности в соответствии с правилами игры*

Даже такие простые игры, как крестики-нолики, являются слишком сложными, чтобы можно было привести в этой книге для них полное дерево игры, поэтому перейдем к описанию тривиальной игры, показанной на рис. 6.2. Возможные коды игрока MAX из корневого узла обозначены как  $a_1$ ,  $a_2$  и  $a_3$ . Возможными ответами на ход  $a_1$  для игрока MIN являются  $b_1$ ,  $b_2$ ,  $b_3$  и т.д. Данная конкретная игра заканчивается после того, как каждый игрок, MAX и MIN, сделают по одному ходу. (Согласно терминологии теории игр, это дерево имеет глубину в один ход и состоит из сделанных двумя участниками ходов, каждый из которых называется **полуходом**.) Полезности терминальных состояний в этой игре находятся в пределах от 2 до 14.



*Рис. 6.2. Дерево игры с двумя полуходами. Узлы ▲ представляют собой “узлы MAX”, в которых очередь хода принадлежит игроку MAX, а узлы ▼ рассматриваются как “узлы MIN”. Терминальные узлы показывают значения полезности для MAX; остальные узлы обозначены их минимаксными значениями. Лучшим ходом игрока MAX от корня является  $a_1$ , поскольку ведет к преемнику с наибольшим минимаксным значением, а наилучшим ответом MIN является  $b_1$ , поскольку ведет к преемнику с наименьшим минимаксным значением*

При наличии дерева игры оптимальную стратегию можно определить, исследуя **минимаксное значение** каждого узла, которое можно записать как  $\text{Minimax-Value}(n)$ . Минимаксным значением узла является полезность (для MAX) пребывания в соответствующем состоянии, при условии, что оба игрока делают ходы оптимальным образом от этого узла и до узла, обозначающего конец игры. Очевидно, что минимаксным значением терминального состояния является просто его полезность. Более того, если есть выбор, игрок MAX должен предпочесть ход, ведущий в состояние с максимальным значением, а игрок MIN — ведущий в состояние с минимальным значением. Поэтому имеет место приведенное ниже соотношение.

$$\text{Minimax-Value}(n) = \begin{cases} \text{Utility}(n), & \text{если } n \text{ — терминальное состояние} \\ \max_{s \in \text{Successors}(n)} \text{Minimax-Value}(s), & \text{если } n \text{ — узел MAX} \\ \min_{s \in \text{Successors}(n)} \text{Minimax-Value}(s), & \text{если } n \text{ — узел MIN} \end{cases}$$

Применим эти определения к дереву игры, показанному на рис. 6.2. Терминальные узлы на низшем уровне уже обозначены числами, которые указывают их полезность. Первый узел MIN, обозначенный как B, имеет трех преемников со значениями 3, 12 и 8, поэтому его минимаксное значение равно 3. Аналогичным образом, другие два узла MIN имеют минимаксное значение, равное 2. Корневым узлом является узел MAX; его преемники имеют минимаксные значения 3, 2 и 2, поэтому сам корневой узел имеет минимаксное значение 3. Можно также определить понятие **минимаксного решения**, принимаемого в корне дерева: действие  $a_1$  является оптимальным выбором для игрока MAX, поскольку ведет к преемнику с наивысшим минимаксным значением.

В этом определении оптимальной игры для игрока MAX предполагается, что игрок MIN также играет оптимальным образом: он максимизирует результат, соответствующий наихудшему исходу игры для MAX. А что было бы, если игрок MIN не играл оптимальным образом? В таком случае можно легко показать (упр. 6.2), что игрок MAX добился бы еще большего. Могут существовать другие стратегии игры против соперников, играющих неоптимальным образом, которые позволяют добиться большего, чем минимаксная стратегия; но эти стратегии обязательно действуют хуже против соперников, играющих оптимально.

## Минимаксный алгоритм

❖ **Минимаксный алгоритм** (листинг 6.1) вычисляет минимаксное решение из текущего состояния. В нем используется простое рекурсивное вычисление минимаксных значений каждого состояния-преемника с непосредственной реализацией определяющих уравнений. Рекурсия проходит все уровни вплоть до листьев дерева, а затем минимаксные значения **резервируются** по всему дереву по мере обратного свертывания рекурсии. Например, в дереве, показанном на рис. 6.2, алгоритм вначале выполнил рекурсию с переходом на нижние уровни до трех нижних левых узлов и применил к ним функцию полезности Utility, чтобы определить, что значения полезности этих узлов соответственно равны 3, 12 и 8. Затем алгоритм определяет минимальное из этих значений, 3, и возвращает его в качестве зарезервированного значения узла *B*. Аналогичный процесс позволяет получить зарезервированные значения 2 для узла *C* и 2 для узла *D*. Наконец, берется максимальное из значений 3, 2 и 2 для получения зарезервированного значения 3 корневого узла.

**Листинг 6.1. Алгоритм вычисления минимаксных решений.** Он возвращает действие, соответствующее наилучшему возможному ходу, т.е. действие, ведущее к результату с наилучшей полезностью, в соответствии с предположением, что противник играет с целью минимизации полезности. Функции **Max-Value** и **Min-Value** используются для прохождения через все дерево игры вплоть до листьев, что позволяет определить зарезервированное значение некоторого состояния

---

```

function Minimax-Decision(state) returns действие action
  inputs: state, текущее состояние игры

  v  $\leftarrow$  Max-Value(state)
  return действие action в множестве Successors(state) со значением v

function Max-Value(state) returns значение полезности
  if Terminal-Test(state) then return Utility(state)
  v  $\leftarrow$   $-\infty$ 
  for a, s in Successors(state) do
    v  $\leftarrow$  Max(v, Min-Value(s))
  return v

function Min-Value(state) returns значение полезности
  if Terminal-Test(state) then return Utility(state)
  v  $\leftarrow$   $\infty$ 
  for a, s in Successors(state) do
    v  $\leftarrow$  Min(v, Max-Value(s))
  return v

```

---

Этот минимаксный алгоритм выполняет полное исследование дерева игры в глубину. Если максимальная глубина дерева равна  $m$  и в каждом состоянии имеются  $b$  допустимых ходов, то времененная сложность этого минимаксного алгоритма составляет  $O(b^m)$ . Для алгоритма, который формирует сразу всех преемников, пространственная сложность равна  $O(bm)$ , а для алгоритма, который формирует преемников по одному, —  $O(m)$  (см. с. 131). Безусловно, в реальных играх такие затраты времени полностью неприемлемы, но данный алгоритм служит в качестве основы математического анализа игр, а также более практических алгоритмов.

### Оптимальные решения в играх с несколькими игроками

Многие популярные игры допускают наличие больше чем двух игроков. Рассмотрим, как можно распространить идею минимаксного алгоритма на игры с несколькими игроками. С точки зрения технической реализации это сделать несложно, но при этом возникают некоторые новые и интересные концептуальные проблемы.

Вначале необходимо заменить единственное значение для каждого узла вектором значений. Например, в игре с тремя игроками, в которой участвуют игроки  $A$ ,  $B$  и  $C$ , с каждым узлом ассоциируется вектор  $\langle v_A, v_B, v_C \rangle$ . Для терминальных состояний этот вектор задает полезность данного состояния с точки зрения каждого игрока. (В играх с двумя игроками и нулевой суммой двухэлементный вектор может быть сокращен до единственного значения, поскольку значения в нем всегда противоположны.) Простейшим способом реализации такого подхода является применение функции *Utility*, которая возвращает вектор значений полезности.

Теперь необходимо рассмотреть нетерминальные состояния. Рассмотрим узел в дереве игры, показанном на рис. 6.3, который обозначен как  $x$ . В этом состоянии игрок  $C$  выбирает, что делать. Два варианта ведут к терминальным состояниям с векторами полезности  $\langle v_A=1, v_B=2, v_C=6 \rangle$  и  $\langle v_A=4, v_B=2, v_C=3 \rangle$ . Поскольку 6 больше чем 3, игрок  $C$  должен выбрать первый ход. Это означает, что если достигнуто состояние  $X$ , то дальнейшая игра приведет к терминальному состоянию с полезностями  $\langle v_A=1, v_B=2, v_C=6 \rangle$ . Следовательно, зарезервированным значением  $X$  является этот вектор. Вообще говоря, зарезервированное значение узла  $n$  представляет собой вектор полезности такого узла-преемника, который имеет наивысшее значение для игрока, выбирающего ход в узле  $n$ .

Любой, кто играет в игры с несколькими игроками, такие как Diplomacy™ (“Дипломатия”), быстро узнает, что в них происходит гораздо больше событий, чем в играх с двумя игроками. В играх с несколькими игроками обычно создаются ~~а~~ альянсы между игроками, либо формальные, либо неформальные. К тому же иногда по мере развития игры альянсы то формируются, то разрушаются. Как можно понять такое поведение? Являются ли альянсы естественным следствием выбора оптимальных стратегий для каждого игрока в игре с несколькими игроками? Как оказалось, они действительно могут стать таким следствием. Например, предположим, что игроки  $A$  и  $B$  имеют слабые позиции, а игрок  $C$  — более сильную позицию. В таком случае и для  $A$ , и для  $B$  часто бывает оптимальным решение атаковать  $C$ , а не друг друга, поскольку иначе  $C$  уничтожит каждого из них по отдельности. Таким образом, сотрудничество становится следствием чисто эгоистичного поведения. Безусловно, как только игрок  $C$  ослабнет под совместным натиском, альянс потеряет свой

смысл и соглашение может нарушить либо игрок *A*, либо игрок *B*. В некоторых случаях явно выраженные альянсы просто становятся конкретным выражением того, что и так произошло бы. А в других случаях попытка нарушить альянс вызывает общественное осуждение, поэтому игроки должны класть на весы немедленно достижимую выгоду от нарушения альянса и долговременный убыток, возникающий из-за того, что их не будут считать заслуживающими доверия. Дополнительная информация об этих сложностях в игре приведена в разделе 17.6.

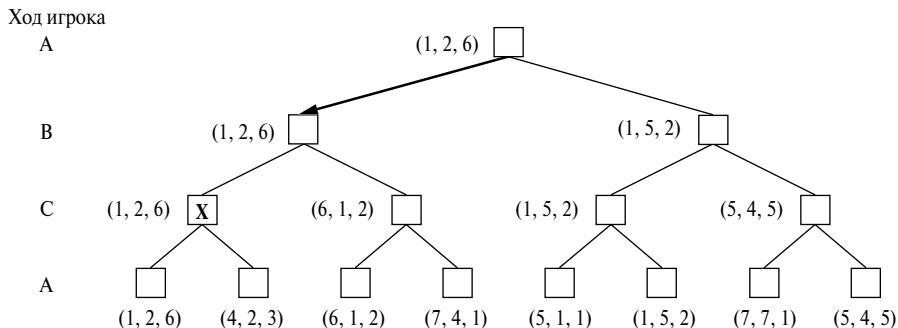


Рис. 6.3. Первые три полухода игры с тремя игроками (*A*, *B*, *C*). Каждый узел обозначен значениями, достижимыми с точки зрения каждого участника. Наилучший ход обозначен стрелкой, исходящей из корня

Если игра не имеет нулевую сумму, то сотрудничество может также возникать даже при наличии всего двух игроков. Допустим, что имеется терминальное состояние с полезностями  $\langle v_A = 1000, v_B = 1000 \rangle$  и что 1000 — максимальная возможная полезность для каждого игрока. В таком случае оптимальная стратегия для обоих игроков заключается в том, чтобы делать все возможное для достижения этого состояния; это означает, что игроки автоматически вступают в сотрудничество для достижения обоядно желаемой цели.

### 6.3. АЛЬФА-БЕТА-ОТСЕЧЕНИЕ

При минимаксном поиске проблема состоит в том, что количество состояний игры, которые должны быть исследованы в процессе поиска, зависит экспоненциально от количества ходов. К сожалению, такую экспоненциальную зависимость устраниТЬ невозможно, но фактически существует возможность сократить ее наполовину. Весь секрет состоит в том, что вычисление правильного минимаксного решения возможно без проверки каждого узла в дереве игры. Это означает, что можно позаимствовать идею **отсечения**, описанную в главе 4, чтобы исключить из рассмотрения большие части дерева. Конкретный метод, рассматриваемый в данной главе, называется **альфа-бета-отсечением**. Будучи применен к стандартному минимаксному дереву, этот метод возвращает такие же ходы, которые вернул бы минимаксный метод, но отсекает ветви, по всей вероятности, не способные повлиять на окончательное решение.

Снова рассмотрим дерево игры с двумя полуходами (см. рис. 6.2) и еще раз проведем расчет оптимального решения, на сей раз обращая особое внимание на то, что известно на каждом этапе этого процесса. Пояснения к данным этапам вычисления

приведены на рис. 6.4. Результат состоит в том, что минимаксное решение можно выявить, даже не приступая к вычислению значений двух листовых узлов.

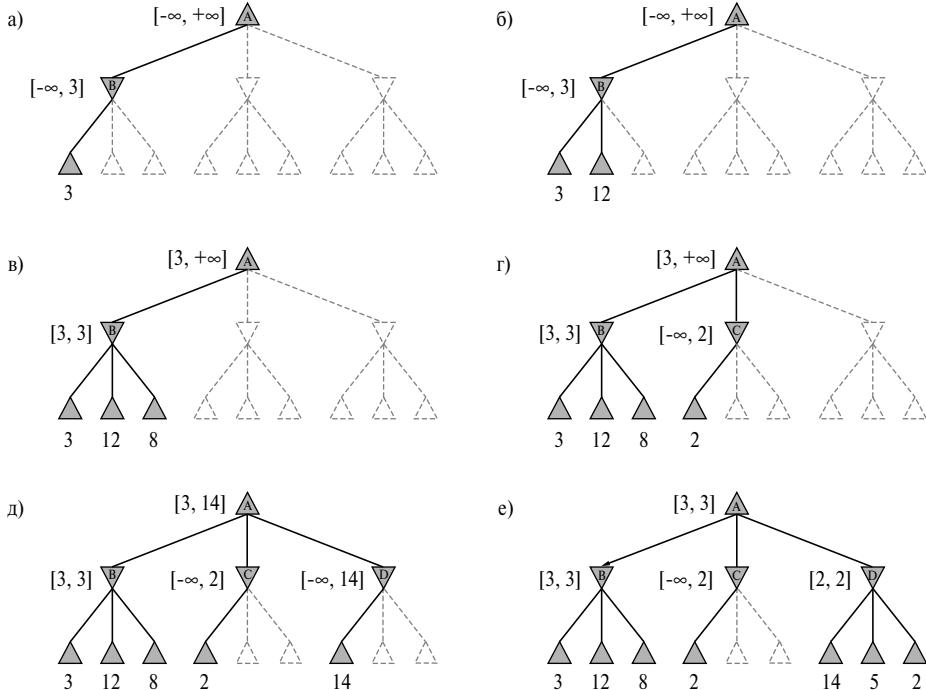


Рис. 6.4. Этапы вычисления оптимального решения для дерева игры, показанного на рис. 6.2; в каждой точке известен ряд возможных значений для каждого узла: первый лист, расположенный ниже узла B, имеет значение 3. Поэтому B, который является узлом MIN, имеет самое большое, значение 3 (а); второй лист, расположенный ниже узла B, имеет значение 12 (б); игрок MIN должен избегать этого хода, поэтому значение B, все еще самое большое, равно 3; третий лист, расположенный ниже узла B, имеет значение 8; мы проверили всех преемников узла B, поэтому значение B в точности равно 3 (в). Теперь можно сделать вывод, что значение корня, самое меньшее, равно 3, поскольку игрок MAX в корне делает выбор со значением 3; первый лист, находящийся ниже C, имеет значение 2. Поэтому C, который представляет собой узел MIN, имеет самое большое, значение 2. Но известно, что узел B позволяет достичь значения 3, поэтому игрок MAX ни в коем случае не должен выбирать узел C. Это означает, что нет смысла проверять остальных преемников узла C. Это — пример применения альфа-бета-отсечения (г). Первый лист, находящийся ниже D, имеет значение 14, поэтому D имеет самое большое, значение 14. Оно все еще выше, чем наилучшая альтернатива для игрока MAX (т.е. 3), поэтому необходимо продолжить исследование преемников узла D. Следует также отметить, что теперь определены предельные значения всех преемников корневого узла, поэтому значение корня также равно, самое большое, 14 (д); второй преемник D имеет значение 5, поэтому снова приходится продолжать исследование. Значение третьего преемника равно 2, поэтому теперь значение D точно равно 2. В корневом узле игрок MAX принимает решение сделать ход, ведущий к узлу B, что позволяет ему получить значение 3 (е)

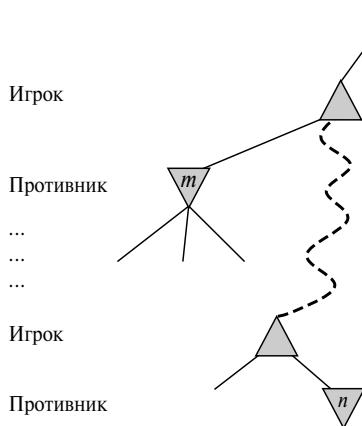
Этот подход может также рассматриваться под другим углом — как упрощение формулы для получения минимаксного значения Minimax-Value. Допустим, что

два преемника узла  $C$  на рис. 6.4, еще не обработанные в процессе вычисления, имеют значения  $x$  и  $y$ , и предположим, что  $z$  — минимальное значение среди  $x$  и  $y$ . В таком случае значение корневого узла можно найти следующим образом:

$$\begin{aligned}\text{Minimax-Value}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \\ &= 3\end{aligned}\quad \text{где } z \leq 2$$

Иными словами, значение корневого узла, а следовательно, и минимаксное решение не зависит от значений отсеченных листовых узлов  $x$  и  $y$ .

Альфа-бета-отсечение может применяться к деревьям любой глубины; к тому же часто возникает возможность отсекать целые поддеревья, а не просто листья. Общий принцип состоит в следующем: рассмотрим узел  $n$ , находящийся где-либо в дереве (рис. 6.5), такой, что участник игры со стороны наблюдателя (назовем его Игрок) имеет возможность выбрать ход, ведущий к этому узлу. Но если Игрок имеет лучший выбор  $m$  либо в родительском узле  $n$ , либо в любой другой точке выбора, находящейся выше в дереве, то ~~если~~ узел  $n$  никогда не будет достигнут в игре, проходящей в действительности. Поэтому после получения достаточной информации об узле  $n$  (путем исследования некоторых из его потомков) для того, чтобы с полной уверенностью прийти к этому заключению, можно выполнить его отсечение.



*Рис. 6.5. Альфа-бета-отсечение: общий случай. Если для Игровка узел  $m$  лучше чем  $n$ , то узел  $n$  никогда не встретится в игре*

Напомним, что минимаксный поиск осуществляется в глубину, поэтому в любой момент времени достаточно рассматривать узлы вдоль единственного пути в дереве. Алгоритм альфа-бета-отсечения получил свое название по следующим двум параметрам, которые представляют пределы в зарезервированных значениях, присутствующих во всех узлах вдоль этого пути:

- $\alpha$  = значение наилучшего варианта (т.е. варианта с самым высоким значением), который был до сих пор найден в любой точке выбора вдоль пути для игрока MAX;

- $\beta$  = значение наилучшего варианта (т.е. варианта с самым низким значением), который был до сих пор найден в любой точке выбора вдоль пути для игрока MIN.

Алгоритм альфа-бета-поиска в процессе своей работы обновляет значения  $\alpha$  и  $\beta$ , а также отсекает оставшиеся ветви в узле (т.е. прекращает рекурсивные вызовы), как только становится известно, что значение текущего узла хуже по сравнению с текущим значением  $\alpha$  или  $\beta$  для игрока MAX или MIN соответственно. Полный алгоритм приведен в листинге 6.2. Рекомендуем читателю проследить за его поведением применительно к дереву, показанному на рис. 6.4.

**Листинг 6.2. Алгоритм альфа-бета-поиска.** Обратите внимание на то, что применяемые здесь процедуры остаются такими же, как и процедуры алгоритма `Minimax`, приведенного в листинге 6.1, за исключением двух строк, введенных как в процедуру `Min-Value`, так и в `Max-Value`, которые сопровождают значения  $\alpha$  и  $\beta$  (а также выполняют соответствующие действия по дальнейшей передаче этих параметров)

---

```

function Alpha-Beta-Search(state) returns действие action
  inputs: state, текущее состояние в игре

  v  $\leftarrow$  Max-Value(state,  $-\infty$ ,  $+\infty$ )
  return действие action из множества Successors(state) со значением v

function Max-Value(state,  $\alpha$ ,  $\beta$ ) returns значение полезности
  inputs: state, текущее состояние в игре
     $\alpha$ , значение наилучшей альтернативы для игрока MAX вдоль
    пути к состоянию state
     $\beta$ , значение наилучшей альтернативы для игрока MIN вдоль
    пути к состоянию state

  if Terminal-Test(state) then return Utility(state)
  v  $\leftarrow$   $-\infty$ 
  for a, s in Successors(state) do
    v  $\leftarrow$  Max(v, Min-Value(s,  $\alpha$ ,  $\beta$ ))
    if v  $\geq \beta$  then return v
     $\alpha \leftarrow \text{Max}(\alpha, v)$ 
  return v

function Min-Value(state,  $\alpha$ ,  $\beta$ ) returns значение полезности
  inputs: state, текущее состояние в игре
     $\alpha$ , значение наилучшей альтернативы для игрока MAX вдоль
    пути к состоянию state
     $\beta$ , значение наилучшей альтернативы для игрока MIN вдоль
    пути к состоянию state

  if Terminal-Test(state) then return Utility(state)
  v  $\leftarrow$   $+\infty$ 
  for a, s in Successors(state) do
    v  $\leftarrow$  Min(v, Max-Value(s,  $\alpha$ ,  $\beta$ ))
    if v  $\leq \beta$  then return v
     $\beta \leftarrow \text{Min}(\beta, v)$ 
  return v

```

---

Эффективность алгоритма альфа-бета-отсечения в высшей степени зависит от того, в каком порядке происходит проверка преемников. Например, на рис. 6.4,  $d$ ,  $e$  невозможно было бы вообще выполнить отсечение каких-либо преемников узла  $D$ , поскольку в первую очередь были бы сформированы наихудшие преемники (с точки зрения игрока MIN). А если бы в первую очередь был сформирован третий преемник, то была бы возможность отсечь двух остальных. На основании этого можно сделать вывод, что имеет смысл стремиться исследовать в первую очередь таких преемников, которые, по всей вероятности, могут стать наилучшими.

Если принять допущение, что это может быть сделано<sup>2</sup>, то окажется, что в алгоритме альфа-бета-отсечения для определения наилучшего хода достаточно исследовать только  $O(b^{m/2})$  узлов, а не  $O(b^m)$  узлов, как при использовании минимаксного алгоритма. Это означает, что эффективный коэффициент ветвления становится равным  $\sqrt{b}$ , а не  $b$ ; например, для шахмат он равен 6, а не 35. Иными словами, за такое же время альфа-бета-поиск позволяет заглянуть в дерево игры примерно в два раза дальше по сравнению с минимаксным поиском. А если исследование преемников происходит в случайном порядке, а не по принципу первоочередного выбора наилучших вариантов, то при умеренных значениях  $b$  общее количество исследованных узлов будет составлять примерно  $O(b^{3m/4})$ . В случае шахмат применение довольно простой функции упорядочения (например, такой, в которой в первую очередь рассматриваются взятия фигур, затем угрозы, затем ходы вперед, а после этого ходы назад) позволяет оставаться в пределах, не превышающих удвоенное значение результата  $O(b^{m/2})$ , который может быть получен в наилучшем случае. Добавление динамических схем упорядочения ходов, в частности, таких, в которых в первую очередь проверяются ходы, обозначенные как наилучшие на предыдущем этапе, позволяют подойти совсем близко к этому теоретическому пределу.

Как было отмечено в главе 3, наличие повторяющихся состояний в дереве поиска может вызвать экспоненциальное увеличение стоимости поиска. В играх повторяющиеся состояния встречаются часто из-за возникновения **транспозиций** — различных перестановок последовательностей ходов, которые оканчиваются в одной и той же позиции. Например, если белые имеют в своем распоряжении ход  $a_1$ , на который черные могут ответить ходом  $b_1$ , а также еще один не связанный с ним ход  $a_2$  на другой стороне доски, на который может быть дан ответ  $b_2$ , то обе последовательности,  $[a_1, b_1, a_2, b_2]$  и  $[a_1, b_2, a_2, b_1]$ , оканчиваются в одной и той же позиции (как и перестановки, начинающиеся с  $a_2$ ). Поэтому целесообразно сохранять оценку каждой конкретной позиции в хэш-таблице при первом ее возникновении, чтобы не приходилось вычислять ее повторно при последующих возникновениях. По традиции хэш-таблица с ранее встретившимися позициями называется **таблицей транспозиций**; она по сути идентична списку *closed* в алгоритме Graph-Search (см. с. 139). Использование таблицы транспозиций может оказаться чрезвычайно эффективное воздействие, которое иногда выражается в удваивании достижимой глубины поиска в шахматах. С другой стороны, если существует возможность вычислять оценки со скоростью в несколько миллионов узлов в секунду, то практически нет смысла хранить данные обо всех этих узлах в таблице транспозиций. Для выбора наиболее ценных из этих узлов были опробованы различные стратегии.

<sup>2</sup> Очевидно, что при этом невозможно достичь идеальных результатов, поскольку в противном случае функцию упорядочения можно было бы использовать для ведения идеальной игры!

## 6.4. НЕИДЕАЛЬНЫЕ РЕШЕНИЯ, ПРИНИМАЕМЫЕ В РЕАЛЬНОМ ВРЕМЕНИ

---

Минимаксный алгоритм формирует все пространство поиска игры, а алгоритм альфа-бета-отсечения позволяет отсекать значительные его части. Тем не менее при использовании алгоритма альфа-бета-отсечения все еще приходится выполнять поиск вплоть до терминальных состояний, по крайней мере, в некоторой области пространства поиска. Обычно задача достижения такой глубины на практике не осуществима, поскольку ходы должны быть сделаны за какое-то приемлемое время, как правило, не больше чем за несколько минут. Вместо этого в статье Шеннона *Programming a computer for playing chess* от 1950 года было предложено, чтобы программы прекращали поиск раньше времени и применяли к состояниям какую-то эвристическую **функцию оценки**, по сути, преобразуя нетерминальные узлы в терминальные листья. Другими словами, в этой статье было предложено модифицировать минимаксный поиск или альфа-бета-поиск в двух отношениях: заменить функцию полезности эвристической функцией оценки `Eval`, которая дает оценку полезности данной позиции, а проверку терминальной позиции заменить ~~проверкой останова~~, которая позволяет решить, когда следует применять функцию `Eval`.

### Функции оценки

Функция оценки возвращает прогноз ожидаемой полезности игры из данной конкретной позиции по аналогии с тем, как эвристические функции, описанные в главе 4, возвращают прогнозируемое значение расстояния до цели. Идея такого “оценщика” в то время, когда Шенон предложил ею воспользоваться, была не нова. В течение многих столетий шахматисты (и поклонники других игр) разработали способы выработки суждений о стоимости позиций, поскольку люди еще более ограничены в объемах поиска, который может быть ими выполнен, чем компьютерные программы. Должно быть очевидно, что производительность любой программы ведения игры зависит от качества применяемой функции оценки. Неточная функция оценки приведет агента к позициям, которые окажутся проигрышными. Поэтому возникает важный вопрос — как именно следует проектировать хорошие функции оценки?

Во-первых, функция оценки должна упорядочивать терминальные состояния таким же образом, как и настоящая функция полезности; в противном случае использующий ее агент может выбрать неоптимальные ходы, даже обладая способностью просчитывать все ходы до конца игры. Во-вторых, вычисления не должны занимать слишком много времени! (В функции оценки можно было бы вызывать `Minimax-Decision` в качестве процедуры и вычислять точную стоимость данной позиции, но это поставило бы под сомнение то, к чему мы стремимся, — экономию времени.) В-третьих, для нетерминальных состояний значения этой функции оценки должны строго коррелировать с фактическими шансами на выигрыш.

Выражение “шансы на выигрыш” на первый взгляд может показаться странным. В конце концов, шахматы — это же не игра с элементами случайности: в ней безусловно известно текущее состояние и для определения следующего хода не нужно бросать жребий. Но если поиск должен прекращаться в нетерминальных состояниях, то в данном алгоритме будет обязательно оставаться неопределенность в отно-

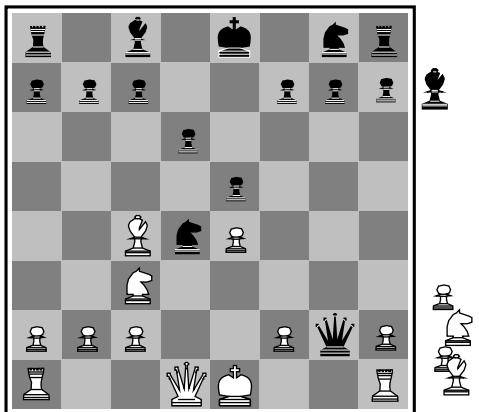
шении окончательных исходов для этих состояний. Неопределенность такого рода вызвана вычислительными, а не информационными ограничениями. Из-за ограниченного объема вычислений, которые разрешено выполнить в функции оценки для данного конкретного состояния, лучшее, что она может сделать, — это принять какое-то предположение в отношении конечного результата.

Рассмотрим эту идею немного более конкретно. Функции оценки чаще всего действуют по принципу вычисления различных **характеристик** данного состояния, например, в шахматах одной из таких характеристик является количество пешек, принадлежащих каждой из сторон. Эти характеристики, вместе взятые, определяют различные категории, или классы эквивалентности состояний: состояния из каждой категории имеют одни и те же значения для всех своих характеристик. Вообще говоря, любая конкретная категория включает некоторые состояния, которые ведут к победе, к ничьей или поражению. Функция оценки не позволяет определить, какими являются те или иные состояния, но способна вернуть единственное значение, которое отражает процентную долю этих состояний в каждом результате. Например, предположим, полученный опыт показывает, что 72% состояний, встретившихся в данной категории, ведут к победе (полезность +1); 20% — к поражению (-1) и 8% к ничьей (0). В таком случае приемлемой оценкой для состояний этой категории становится взвешенное среднее, или **ожидаемое значение**:  $(0.72 \times +1) + (0.20 \times -1) + (0.08 \times 0) = 0.52$ . В принципе, ожидаемое значение можно определить для каждой категории, получив в итоге функцию оценки, применимую для любого состояния. Как и в случае терминальных состояний, функция оценки не обязана возвращать фактические ожидаемые значения, при условии, что упорядочение состояний остается тем же самым.

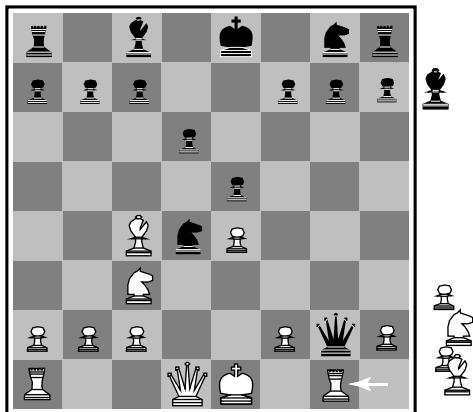
На практике для проведения анализа такого рода требуется учитывать слишком много категорий и поэтому накопить слишком много опыта, чтобы можно было оценить все вероятности выигрыша. Вместо этого в большинстве функций оценки вычисляются отдельные представленные в числовом виде значения вклада, зависящего от каждой характеристики, после чего эти значения комбинируются для поиска суммарного значения. Например, в учебниках по шахматам для начинающих можно найти приближенные оценки **стоимости материала** для каждой фигуры: например, такие, что пешка имеет стоимость 1, конь или слон — 3, ладья — 5, а ферзь — 9. Другие характеристики, такие как “хорошая пешечная структура” и “безопасность короля”, могут оцениваться как равные, скажем, половине стоимости пешки. После этого стоимости таких характеристик просто складываются для получения оценки позиции. Надежное преимущество, эквивалентное стоимости пешки, расценивается как значительная вероятность выигрыша, а надежное преимущество в три пешки должно почти наверняка обеспечить победу, как показано на рис. 6.6, а. В математике функция оценки такого типа называется **взвешенной линейной функцией**, поскольку она может быть представлена следующим образом:

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s)$$

где каждый коэффициент  $w_i$  представляет собой вес, а каждая функция  $f_i$  оценивает некоторую характеристику позиции. В шахматах функция  $f_i$  может определять количество на доске фигур каждого вида, а коэффициент  $w_i$  — оценивать стоимости этих фигур (1 за пешку, 3 за слона и т.д.).



а) Ход белых



б) Ход белых

*Рис. 6.6. Две немного разные шахматные позиции: черные имеют преимущество в одного коня и двух пешек и должны выиграть партию (а); черные проигрывают после того, как белые берут ферзя (б)*

На первый взгляд метод вычисления суммы стоимостей характеристик может показаться приемлемым, но в действительности он основан на очень радикальном допущении, что вклад каждой характеристики не зависит от стоимости других характеристик. Например, присваивая слону стоимость 3, мы игнорируем тот факт, что слоны становятся более мощными в конце игры, когда имеют большой объем пространства для маневра. По этой причине в современных программах для шахмат и других игр используются также нелинейные комбинации характеристик. Например, пара слонов может стоить немного больше по сравнению с удвоенной стоимостью одного слона, а слон стоит немного больше в конце игры, чем в начале.

Внимательный читатель должен был заметить, что все эти характеристики и веса не входят в состав шахматных правил! Они были выработаны в течение столетий на основе опыта игры людей в шахматы. Применение этих характеристик и весов на основе описанной линейной формы оценки позволяет добиться наилучшей аппроксимации по отношению к истинному упорядочению состояний по стоимости. В частности, опыт показывает, что надежное материальное преимущество больше чем в один пункт, по всей вероятности, приводит к выигрышу при всех прочих равных условиях; преимущество в три пункта является достаточным почти для безусловной победы. В таких играх, где опыт указанного вида отсутствует, веса функции оценки могут быть получены с помощью методов машинного обучения, приведенных в главе 18. Является обнадеживающим тот факт, что применение указанных методов к шахматам подтвердило, что слон действительно имеет стоимость, примерно равную трем пешкам.

### Прекращение поиска

Следующий этап состоит в том, что алгоритм Alpha-Beta-Search должен быть модифицирован так, чтобы он вызывал эвристическую функцию *Eval*, когда возникает необходимость остановить поиск. С точки зрения реализации необходимо

заменить две строки в листинге 6.2, в которых упоминается функция Terminal-Test, следующей строкой:

```
if Cutoff-Test(state, depth) then return Eval(state)
```

Необходимо также предусмотреть выполнение определенных технических операций для того, чтобы текущее значение глубины *depth* наращивалось при каждом рекурсивном вызове. Наиболее прямолинейный подход к управлению объемом поиска состоит в том, что должен устанавливаться фиксированный предел глубины, чтобы функция Cutoff-Test (*state, depth*) возвращала значение *true* при всех значениях *depth*, превышающих некоторую фиксированную глубину *d*. (Она должна также возвращать *true* для всех терминальных состояний, как было предусмотрено и в функции Terminal-Test.) Глубина *d* выбрана таким образом, чтобы используемое время не превышало допустимое по правилам игры.

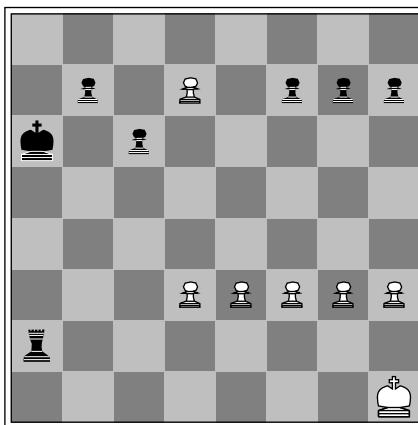
Более надежный подход состоит в использовании метода итеративного углубления, который определен в главе 3. По окончании отведенного времени программа возвращает ход, выбранный по итогам наиболее глубокого завершенного поиска. Однако подобные подходы могут приводить к ошибкам, обусловленным приближенным характером функции оценки. Еще раз рассмотрим простую функцию оценки для шахмат, основанную на учете преимущества в материале. Предположим, что программа выполняет поиск до предела глубины, достигая позиции, приведенной на рис. 6.6, б, где черные имеют перевес на одного коня и две пешки. Программа сообщила бы об этом как об эвристическом значении данного состояния, объявив тем самым, что это состояние, по всей вероятности, приведет к победе черных. Но на следующем ходу белые берут ферзя черных без компенсации. Поэтому в действительности данная позиция является выигрышной для белых, но об этом можно было бы узнать, только заглянув вперед еще на один полуход.

Очевидно, что требуется более сложная проверка останова. Функция оценки должна применяться только к позициям, которые являются **спокойными**, т.е. характеризующимися низкой вероятностью того, что в них в ближайшем будущем произойдут резкие изменения в стоимости. Например, в шахматах такие позиции, в которых могут быть сделаны желательные взятия фигур, не являются спокойными для такой функции оценки, в которой учитывается лишь материал. Неспокойные позиции могут быть дополнительно развернуты до тех пор, пока не будут достигнуты спокойные позиции. Подобный дополнительный поиск называется **поиском спокойных позиций**; иногда он ограничивается тем, что в нем рассматриваются только ходы определенных типов, такие как ходы со взятием фигур, что позволяет быстро устранять все неопределенности в этой позиции.

Задача устранения **эффекта горизонта** является более сложной. Этот эффект возникает, если программа сталкивается с каким-то ходом противника, который причиняет серьезный ущерб и в конечном итоге является неизбежным. Рассмотрим шахматную позицию, приведенную на рис. 6.7. Черные превосходят белых по количеству материала, но если белые смогут продвинуть свою пешку с седьмой горизонтали на восьмую, то пешка станет ферзем и обеспечит легкую победу белых. Черные могут отсрочить этот итог на 14 полуходов, объявляя шах белым с помощью ладьи, но пешка неизбежно станет ферзем. Одним из недостатков поиска на фиксированную глубину является то, что применяемый при

этом алгоритм не позволяет определить, что такие отвлекающие ходы не способны предотвратить ход с превращением пешки в ферзя. В этом случае принято считать, что отвлекающие ходы выводят неизбежный ход с превращением пешки в ферзя “за пределы горизонта поиска” — в то место, где опасный ход невозможно обнаружить.

По мере того как совершенствование аппаратных средств, применяемых для ведения игры в шахматы, приводит к увеличению глубины поиска, становится все более возможным то, что эффект горизонта будет возникать не так часто, поскольку очень длинные последовательности ходов, позволяющие отсрочить выполнение нежелательного хода, возникают крайне редко. Для предотвращения эффекта горизонта без слишком значительного увеличения стоимости поиска оказалось также весьма эффективным использование **одинарных расширений**. Одинарным расширением называется ход, который “безусловно лучше” по сравнению со всеми другими ходами в данной конкретной позиции. Поиск с одинарным расширением позволяет выйти за обычные пределы глубины поиска без внесения значительных издержек, поскольку в нем коэффициент ветвления равен 1. (Поиск спокойной позиции может рассматриваться как один из вариантов одинарных расширений.) На рис. 6.7 поиск с одинарным расширением позволяет найти выполняемый в конечном итоге ход превращения пешки в ферзя, при условии, что ходы черных с объявлением шаха и ходы белых королем могут быть определены как “безусловно лучшие” по сравнению с другими вариантами.



Ход черных

*Рис. 6.7. Проявление эффекта горизонта. Серия шахов черной ладьей приводит к принудительному выводу неизбежного хода белых с превращением пешки в ферзя “за горизонт”, в связи с чем эта позиция начинает казаться допускающей выигрыши черных, тогда как фактически она является выигрышной для белых*

До сих пор речь шла о том, что прекращение поиска на определенном уровне и выполнение альфа-бета-отсечения, по-видимому, не влияет на результат. Существует также возможность выполнять **предварительное отсечение**, а это означает, что некоторые ходы в данном конкретном узле отсекаются немедленно,

без дальнейшего рассмотрения. Очевидно, что большинство людей, играющих в шахматы, рассматривают лишь несколько ходов из каждой позиции (по крайней мере, сознательно). К сожалению, этот подход является довольно опасным, поскольку нет никакой гарантии того, что не произойдет отсечение лучшего хода. А если отсечение применяется недалеко от корня, результат может оказаться катастрофическим, поскольку слишком часто возникают такие ситуации, что программа пропускает некоторые “очевидные” ходы. Предварительное отсечение может использоваться безопасно в особых ситуациях (например, если два хода являются симметричными или эквивалентными по каким-то другим признаками, то необходимо рассматривать только один из них) или при анализе узлов, которые находятся глубоко в дереве поиска.

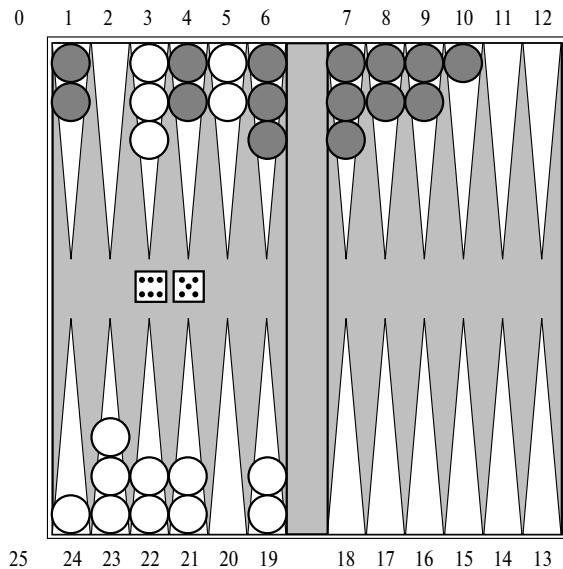
В результате совместного использования всех методов, описанных выше, появляется возможность создать программу, которая неплохо играет в шахматы (или другие игры). Предположим, что реализована функция оценки для шахмат, предусмотрена разумная проверка останова в сочетании с поиском спокойной позиции, а также предусмотрена большая таблица транспозиций. Кроме того, предположим, что, затратив целые месяцы на скрупулезную разработку программ, функционирующих на уровне битов, мы получили возможность формировать и оценивать примерно миллион узлов в секунду на новейшем персональном компьютере, что позволяет выполнять поиск приблизительно среди 200 миллионов узлов в расчете на каждый ход при стандартном контроле времени (три минуты на каждый ход). Коэффициент ветвления для шахмат составляет в среднем примерно 35, а  $35^5$  равно приблизительно 50 миллионам, поэтому при использовании минимаксного поиска мы получим возможность заглядывать вперед лишь приблизительно на пять полуходов. Такая программа, хотя и не совсем некомпетентная, может быть легко обманута человеком, игроком в шахматы среднего уровня, который иногда способен планировать на шесть или восемь полуходов вперед. Альфа-бета-поиск позволяет достичь глубины приблизительно в 10 полуходов, что приводит к усилению игры до уровня мастера. В разделе 6.7 описаны дополнительные методы отсечения, которые позволяют увеличить эффективную глубину поиска примерно до 14 полуходов. Чтобы достичь уровня гроссмейстера, требуется тщательно настроенная функция оценки и большая база данных с записями оптимальных ходов в дебюте и эндшпиле. Не мешало бы также иметь суперкомпьютер для эксплуатации на нем такой программы!

## 6.5. ИГРЫ, КОТОРЫЕ ВКЛЮЧАЮТ ЭЛЕМЕНТ СЛУЧАЙНОСТИ

В реальной жизни происходит много непредсказуемых внешних событий, из-за которых люди попадают в непредвиденные ситуации. Эта непредсказуемость отражается во многих играх за счет включения элемента случайности, такого как мечение жребия. Таким образом, игры с элементом случайности позволяют нам на один шаг приблизиться к реальности и поэтому имеет смысл рассмотреть, как это отражается на процессе принятия решений.

Одной из типичных игр, в которых сочетается удача и искусство, являются наряды. Перед каждым своим ходом игрок бросает кубики для определения допустимых

ходов. Например, в позиции игры в нарды, приведенной на рис. 6.8, белым выпали очки 6–5 и они имеют четыре возможных хода.



*Рис. 6.8. Типичная позиция игры в нарды. Цель игры состоит в том, чтобы снять с доски все фишкы. Белые ходят по часовой стрелке к полю 0, а черные — против часовой стрелки к полю 0. Фишка может переходить на любое поле, если на нем не находится несколько фишек противника; если же на этом поле есть только одна фишка противника, она попадает в плен и должна начать свое движение с самого начала. В показанной здесь позиции белые после метания жребия получили очки 6–5 и должны выбирать среди четырех допустимых ходов: (5–10, 5–11), (5–11, 19–24), (5–10, 10–16) и (5–11, 11–16)*

Хотя белым известно, каковы их допустимые ходы, они не знают, какие очки принесет метание жребия черным, и поэтому не могут определить, какими будут допустимые ходы черных. Это означает, что белые не имеют возможности сформировать стандартное дерево игры такого типа, которое встретилось нам в шахматах, а также крестиках-ноликах. Дерево игры в нарды, кроме узлов MAX и MIN, должно включать **узлы жеребьевки**. Узлы жеребьевки обозначены на рис. 6.9 кружками. Ветви, ведущие от каждого узла жеребьевки, обозначают возможные результаты метания жребия, поэтому каждая из них отмечена надписью с указанием количества очков и вероятности, с которой могут быть получены эти очки. Существуют 36 различных сочетаний очков на двух кубиках, и все они являются равновероятными; но поскольку такие сочетания очков, как 6–5 и 5–6, являются одинаковыми, имеется только 21 различимое сочетание очков. Вероятность появления шести дублей (от 1–1 до 6–6) равна 1/36, а каждое из 15 остальных различных сочетаний очков имеет вероятность 1/18.

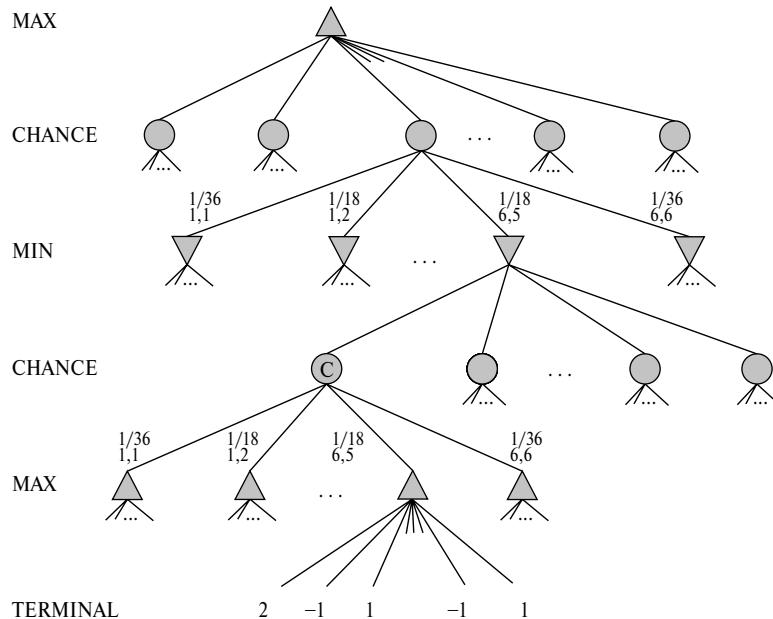


Рис. 6.9. Схематическое изображение дерева игры для одной из позиций игры в нарды

Следующий этап состоит в том, чтобы понять, как следует принимать правильные решения. Безусловно, и в этом случае требуется найти такой ход, который ведет к наилучшей позиции. Однако результирующие позиции не имеют определенных минимаксных значений. Вместо этого существует возможность вычислить только **ожидаемое значение**, в котором ожидаемый результат устанавливается с учетом всех возможных выпадений жребия, которые могут произойти. Это приводит к обобщению **минимаксного значения** для детерминированных игр до ~~ожидаемого минимаксного значения~~ (expectiminimax value) для игр с узлами жеребьевки. Терминальные узлы и узлы MAX и MIN (для которых известны результаты жеребьевки) применяются точно так же, как и прежде, а узлы жеребьевки оцениваются путем получения взвешенного среднего значений, полученных в результате всех возможных выпадений жребия, т.е. следующим образом:

$$\text{Expectiminimax}(n) = \begin{cases} \text{Utility}(n), & \text{если } n \text{ - терминальное состояние} \\ \max_{s \in \text{Successors}(n)} \text{Expectiminimax}(s), & \text{если } n \text{ - узел MAX} \\ \min_{s \in \text{Successors}(n)} \text{Expectiminimax}(s), & \text{если } n \text{ - узел MIN} \\ \sum_{s \in \text{Successors}(n)} P(s) \cdot \text{Expectiminimax}(s), & \text{если } n \text{ - узел жеребьевки} \end{cases}$$

где функция определения преемника для узла жеребьевки  $n$  просто дополняет состояние  $n$  каждым возможным выпадением жребия для формирования каждого преемника  $s$ , а  $P(s)$  — вероятность, с которой происходит выпадение жребия. Результаты вычисления этих уравнений могут резервироваться рекурсивно во всех узлах вплоть до корня дерева точно так же, как и в минимаксном алгоритме. Оставляем читателю проработку всех деталей этого алгоритма в качестве упражнения.

### Оценка позиции в играх с узлами жеребьевки

По аналогии с минимаксными значениями, очевидный подход к использованию ожидаемых минимаксных значений состоит в том, чтобы останавливать поиск в некоторой точке и применять функцию оценки к каждому листу. На первый взгляд может показаться, что функции оценки для таких игр, как нарды, должны быть полностью подобными функциям оценки для шахмат, ведь от них требуется лишь то, чтобы они присваивали более высокие оценки лучшим позициям. Но в действительности наличие узлов жеребьевки означает, что требуется более тщательный анализ смысла таких оценочных значений. На рис. 6.10 показано, что происходит в играх с элементами случайности — при использовании функции оценки, которая присваивает листьям значения  $[1, 2, 3, 4]$ , наилучшим является ход  $A_1$ , а если присваиваются значения  $[1, 20, 30, 400]$ , наилучшим становится ход  $A_2$ . Поэтому программа ведет себя полностью по-разному, если вносятся изменения в шкалу некоторых оценочных значений! Как оказалось, такой чувствительности к изменению шкалы можно избежать при условии, что функция оценки представляет собой положительную линейную трансформацию вероятности выигрыша из некоторой позиции (или, в более общем смысле, ожидаемой полезности данной позиции). В этом состоит важное и общее свойство ситуаций, связанных с наличием неопределенности, и это свойство дополнительно рассматривается в главе 16.

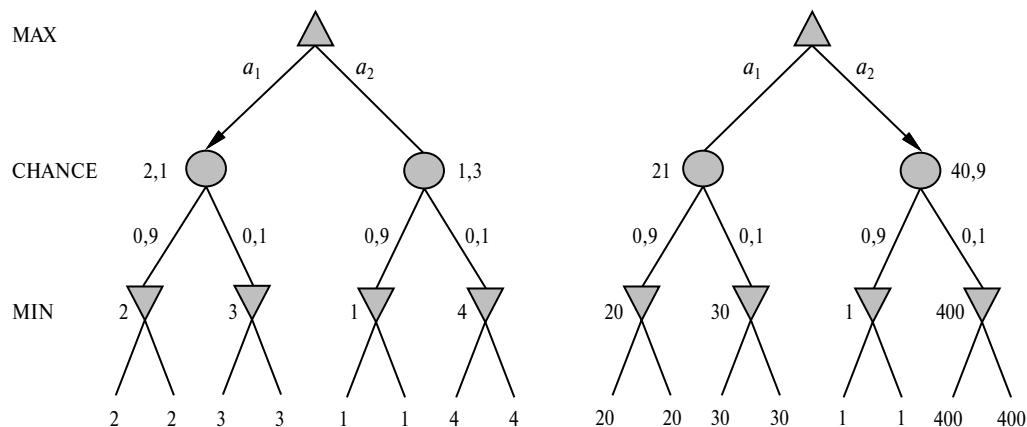


Рис. 6.10. Пример того, что в результате трансформации оценочных значений листьев, при которой не изменяется упорядочение этих значений, наилучший ход становится другим

## Сложность оценки ожидаемых минимаксных значений

Если бы в программе были заранее известны все выпадения жребия, которые должны произойти в течение остальной части игры, то поиск решения в игре с жеребьевкой был бы полностью аналогичным поиску решения в игре без жеребьевки, который осуществляется минимаксным алгоритмом поиска за время  $O(b^m)$ . Но поскольку в алгоритме, использующем ожидаемые минимаксные значения, рассматриваются также все возможные последовательности выпадения жребия, для его работы требуется время  $O(b^m n^m)$ , где  $n$  — количество различных вариантов выпадения жребия.

Даже если глубина поиска ограничена некоторой небольшой величиной  $d$ , из-за таких дополнительных затрат времени, намного более значительных по сравнению с минимаксным алгоритмом, в большинстве игр с элементами случайности становится неосуществимым стремление заглянуть в дерево поиска на очень большую глубину. В нарядах  $n$  равно 21, а  $b$  обычно составляет приблизительно 20, но в некоторых ситуациях может достигать величины 4000 для выпадений жребия, включающих повторяющиеся очки. По-видимому, все, на что можно рассчитывать, — это заглянуть вперед на три полухода.

Еще один способ размышлении об этой проблеме состоит в следующем: преимуществом альфа-бета-поиска является то, что в нем игнорируются будущие направления развития игры, которые просто не должны быть реализованы, если в игре всегда выбирается наилучший ход. Поэтому альфа-бета-поиск сосредоточивается на наиболее вероятных событиях. В играх с жеребьевкой вероятных последовательностей ходов не может быть, поскольку, для того, чтобы были сделаны данные ходы, вначале должен выпасть правильный жребий, который сделал бы их допустимыми. В этом заключается общая проблема, возникающая в любой такой ситуации, когда в картину мира вмешивается неопределенность: количество вариантов дальнейших действий становится бессмысленным, поскольку мир, скорее всего, будет играть совсем в другую игру.

Несомненно, читателю уже приходила в голову мысль, что к деревьям игр с узлами жеребьевки, по-видимому, можно было бы применить какой-то метод, подобный альфа-бета-отсечению. Как оказалась, такая возможность действительно существует. Анализ узлов MIN и MAX остается неизменным, но можно также обеспечить отсечение узлов жеребьевки с использованием некоторой доли изобретательности. Рассмотрим узел жеребьевки  $C$ , приведенный на рис. 6.9, и определим, что будет происходить со значением этого узла по мере исследования и оценки его дочерних узлов. Можно ли найти верхний предел значения  $C$  до того, как будут проверены все его дочерние узлы? (Напомним, что именно это требуется в альфа-бета-поиске для того, чтобы можно было выполнить отсечение узла и его поддерева.) На первый взгляд такое требование может показаться невыполнимым, поскольку значение узла  $C$  представляет собой среднее значение его дочерних узлов. А до тех пор пока не будут рассмотрены все выпадения жребия, это среднее может представлять собой что угодно, поскольку сами неисследованные дочерние узлы могут иметь вообще любое значение. Тем не менее, если будут установлены пределы допустимых значений функции полезности, то появится возможность определять пределы и этого среднего. Например, если будет установлено, что все значения полезности находятся в пределах от

+3 до -3, то значения листовых узлов становятся ограниченными, а это, в свою очередь, позволяет установить верхний предел значения узла жеребьевки, не рассматривая все его дочерние узлы.

## Карточные игры

Карточные игры интересны не только тем, что они часто бывают связаны с денежными ставками, но и по многим другим причинам. Количество различных вариантов карточных игр чрезвычайно велико, но в данной главе мы сосредоточимся на таких вариантах, в которых карты тасуют случайным образом в начале игры и каждый игрок получает на руки карты, которые он не показывает другим игрокам. К таким играм относятся бридж, вист, “Черви” и некоторые виды покера.

На первый взгляд может показаться, что карточные игры полностью аналогичны играм с жеребьевкой: раздача карт происходит случайным образом, а сами карты определяют, какие ходы могут быть сделаны каждым игроком. Однако в картах вся жеребьевка происходит с самого начала! Это замечание будет обсуждаться ниже более подробно и будет показано, что такая особенность рассматриваемых карточных игр весьма полезна на практике. Вместе с тем, это замечание одновременно является совершенно неправильным по очень интересным причинам.

Представьте себе, что два игрока, MAX и MIN, разыгрывают какую-то реальную раздачу по четыре карты в бриdge с двумя игроками, в котором открыты все карты. На руках находятся следующие карты, и игрок MAX должен ходить первым:

MAX: ♠ 6 ♦ 6 ♣ 9 8      MIN: ♥ 4 ♠ 2 ♣ 10 5

Предположим, что игрок MAX ходит с карты ♣ 9. Теперь должен ходить игрок MIN, который может выбросить карту ♣ 10 или ♣ 5. Игрок MIN кладет карту ♣ 10 и забирает взятку. Затем очередь хода переходит к игроку MIN, который ходит картой ♠ 2. У игрока MAX масти пика нет (и поэтому он не может забрать эту взятку), следовательно, он обязан выбросить какую-то другую карту. Очевидным вариантом является карта ♦ 6, поскольку две другие оставшиеся карты являются старшими. Теперь, какой бы картой не ходил игрок MIN во время следующего розыгрыша, игрок MAX возьмет две оставшиеся взятки и игра окончится ничьей, при двух взятках у каждого игрока. С использованием подходящего варианта минимаксного поиска (см. упр. 6.12) можно легко показать, что фактически ход картой ♣ 9, сделанный игроком MAX, был оптимальным.

Теперь заменим карты на руках игрока MIN и вместо карты ♥ 4 введем карту ♦ 4:

MAX: ♥ 6 ♦ 6 ♣ 9 8      MIN: ♦ 4 ♠ 2 ♣ 10 5

Рассматриваемые два случая являются полностью симметричными: ход игры будет одинаковым, за исключением того, что при розыгрыше второй взятки игрок MAX выбросит карту ♥ 6. Игра снова окончится ничьей при двух взятках у каждого игрока, и ход картой ♣ 9 является оптимальным.

До сих пор все шло хорошо. А теперь скроем одну из карт игрока MIN и допустим, что игрок MAX знает, что у игрока MIN на руках либо первая раздача (с картой ♥ 4), либо вторая раздача (с картой ♦ 4), но он не знает, какая именно из них. Игрок MAX рассуждает следующим образом.

Ход картой ♦ 9 является оптимальным решением в игре против первой и второй раздачи на руках игрока MIN, поэтому теперь этот ход должен быть оптимальным, поскольку известно, что на руках у игрока MIN имеется один из этих двух вариантов раздачи.

На более обобщенном уровне можно сказать, что игрок MAX использует подход, который может быть назван “усреднением по прогнозам”. Идея его состоит в том, чтобы при наличии карт на руках у противника, которые не видны игроку, оценивать каждый возможный вариант действий, вначале вычисляя минимаксное значение этого действия применительно к каждой возможной раздаче карт, а затем вычисляя ожидаемое значение по всем раздачам с использованием вероятности каждой раздачи.

Если читатель посчитает такой подход разумным (или если он не может судить о нем, поскольку не знаком с бриджем), то ему следует поразмыслить над приведенным ниже рассказом.

**Первый день.** Дорога A ведет к куче золотых слитков; дорога B ведет к развилке. Если вы от развилки пойдете налево, то найдете гору драгоценностей, а если пойдете направо, то попадете под автобус.

**Второй день.** Дорога A ведет к куче золотых слитков; дорога B ведет к развилке. Если вы от развилки пойдете направо, то найдете гору драгоценностей, а если пойдете налево, то попадете под автобус.

**Третий день.** Дорога A ведет к куче золотых слитков; дорога B ведет к развилке. Если вы от развилки выберете правильное направление, то найдете гору драгоценностей, а если неправильное, то попадете под автобус.

Очевидно, что в первые два дня решение выбрать дорогу B не лишено смысла, но в третий день ни один человек в здравом уме не выберет дорогу B. Однако именно такой вариант подсказывает метод усреднения по прогнозам: дорога B является оптимальной в ситуациях, возникающих в первый и второй дни, поэтому она оптимальна и в третий день, поскольку должна иметь место одна из двух предыдущих ситуаций. Вернемся к карточной игре: после того как игрок MAX пойдет картой ♦ 9, игрок MIN заберет взятку картой ♦ 10. Как и прежде, MIN пойдет с карты ♠ 2, но теперь игрок MAX окажется перед развилкой на дороге без каких-либо указаний. Если игрок MAX выбросит карту ♥ 6, а у игрока MIN все еще будет оставаться карта ♥ 4, то эта карта ♥ 4 станет козырной и игрок MAX проиграет игру. Аналогичным образом, если игрок MAX выбросит карту ♦ 6, а у игрока MIN все еще будет оставаться карта ♦ 4, игрок MAX также проиграет. Поэтому игра с первым ходом картой ♦ 9 ведет к ситуации, в которой игрок MAX имеет 50%-ную вероятность проигрыша. (Для него было бы гораздо лучше вначале сыграть картами ♥ 6 и ♦ 6, гарантируя для себя ничейную игру.)

Из всего этого можно извлечь урок, что при недостатке информации игрок должен учитывать, какую информацию он будет иметь в каждый момент игры. Недостаток алгоритма, применяемого игроком MAX, состоит в том, что в нем предполагается, будто при каждой возможной раздаче игра будет развиваться так, как если бы все карты оставались видимыми. Как показывает данный пример, это вынуждает игрока MAX действовать таким образом, как будто неопределенность разрешится, когда настанет время. Кроме того, в алгоритме игрока MAX никогда не принимается решение, что нужно собирать информацию (или предоставлять информацию партнеру), поскольку этого не требуется делать в рамках каждой отдельной раздачи; тем не менее в таких играх, как бридж, часто бывает целесообразно сыграть такой кар-

той, которая помогла бы кое-что узнать о картах противника или сообщить партнеру о своих собственных картах. Такие стереотипы поведения формируются автоматически оптимальным алгоритмом ведения игр с неполной информацией. Подобный алгоритм выполняет поиск не в пространстве состояний мира (под этим подразумеваются карты, находящиеся на руках у игроков), а в пространстве **доверительных состояний** (представлений о том, кто какие карты имеет и с какими вероятностями). Авторы смогут объяснить этот алгоритм должным образом в главе 17 после разработки всего необходимого вероятностного инструментария. А в данной главе необходимо также остановиться на одном заключительном и очень важном замечании: в играх с неполной информацией лучше всего выдавать противнику как можно меньше информации, а наилучший способ сделать это чаще всего состоит в том, чтобы действовать непредсказуемо. Вот почему санитарные врачи, посещая для проверки предприятия общественного питания, не сообщают заранее о своих визитах.

## 6.6. СОВРЕМЕННЫЕ ИГРОВЫЕ ПРОГРАММЫ

---

Публичная демонстрация программ ведения игр представляет собой для искусственного интеллекта примерно то же, что и участие в автогонках международного масштаба для автомобильной промышленности — суперсовременные игровые программы действуют поразительно быстро, чрезвычайно тщательно настроенные компьютеры воплощают в себе наиболее качественные инженерные решения, но это все не предназначено для рядового покупателя. А некоторые исследователи даже считают, что ведение игр представляет собой нечто, не имеющее отношения к основному направлению развития искусственного интеллекта. Тем не менее эта область продолжает порождать не только всеобщее восхищение, но и постоянный поток инноваций, которые затем усваиваются более широким сообществом разработчиков.

❖ **Шахматы.** В 1957 году Герберт Саймон предсказал, что через 10 лет компьютеры победят человека — чемпиона мира по шахматам. Через сорок лет программа Deep Blue победила Гарри Каспарова в показательном матче из шести игр. Саймон ошибся, но лишь с коэффициентом 4. Каспаров писал:

Решающей игрой в этом матче была вторая партия, которая оставила глубокий след в моей памяти... Мы наблюдали события, которые намного превосходили самые невероятные ожидания в отношении того, насколько хорошо компьютер будет способен предвидеть долговременные позиционные последствия своих решений. Машина отказалась перейти в позицию, имеющую явное, но кратковременное преимущество, продемонстрировав вполне человеческое ощущение опасности [774].

Программа Deep Blue была создана Мьюрреем Кэмпбеллом, Фенгсунг Су и Джозефом Хоаном из компании IBM [216] на основе проекта Deep Thought, разработанного ранее Кэмпбеллом и Су в университете Карнеги-Меллона (Carnegie-Mellon University — CMU). Компьютер-победитель представлял собой параллельный компьютер с 30 процессорами IBM RS/6000. На этом компьютере эксплуатировались средства “программного поиска” и 480 специализированных СБИС шахматных процессоров, которые осуществляли выработку ходов (включая упорядочение ходов), “аппаратного поиска” для последних нескольких уровней дерева и проводилась оценка листовых узлов. В программе Deep Blue в среднем осуществлялся поиск 126 миллионов узлов в секунду, а пиковая скорость достигала 330 миллионов узлов в

секунду. Эта программа формировалась вплоть до 30 миллиардов позиций в расчете на каждый ход, обычно достигая глубины поиска 14. Основой этой программы является стандартный альфа-бета-поиск с итеративным углублением на основе таблицы транспозиций, но ключом к успеху этой программы, по-видимому, стала ее способность вырабатывать расширения, выходящие за пределы глубины поиска для достаточно интересных линий форссирующих/форсированных ходов. В некоторых случаях этот поиск достигал глубины в 40 полуходов. Функция оценки охватывала свыше 8000 характеристик, причем многие из них описывали в высшей степени специфичные шаблоны расположения фигур. Использовался справочник дебютов, состоящий примерно из 4000 позиций, а также база данных с 700 000 игр гроссмейстеров, из которой программа могла извлекать согласованные рекомендации. Кроме того, в этой системе применялась большая база данных эндшпилей, состоящая из позиций с решениями, в которой содержались все позиции с пятью фигурами и многие позиции с шестью фигурами. Использование этой базы данных привело к значительному увеличению эффективной глубины поиска, что позволило программе Deep Blue в некоторых случаях играть идеально даже за много ходов до матта.

Успех программы Deep Blue укрепил и без того широко распространенное мнение, что прогресс в области ведения компьютерных игр достигается главным образом за счет все более мощного аппаратного обеспечения; к тому же распространение таких взглядов стимулировалось компанией IBM. Создатели программы Deep Blue, с другой стороны, утверждают, что важную роль сыграло также расширение поиска и применение продуманной функции оценки [216]. Более того, нам известно, что некоторые новейшие алгоритмические усовершенствования позволяют программам, работающим на стандартных персональных компьютерах, побеждать на каждом мировом чемпионате по компьютерным шахматам с 1992 года и часто наносить поражение противникам с массовой параллельной архитектурой, способным выполнять поиск в 1000 раз большего количества узлов в секунду. Для сокращения эффективного коэффициента ветвления меньше чем до 3 (по сравнению с фактическим коэффициентом ветвления, составляющим около 35) применяются всевозможные эвристики отсечения. Наиболее важной из них является эвристика с **пустым ходом**, в которой вырабатывается хороший нижний предел значения позиции с использованием поверхностного поиска, в котором противнику в начале игры разрешается сделать ход дважды. Этот нижний предел часто позволяет выполнять альфа-бета-отсечение без затрат на полный поиск в глубину. Является также важным метод **отсечения ненужных ходов**, который позволяет решить заранее, какие ходы вызовут альфа-бета-отсечение в узлах-преемниках.

Группа разработчиков Deep Blue отказалась воспользоваться шансом провести матч-реванш, предложенный Каспаровым. Вместо этого в одном из самых последних крупных соревнований в 2002 году против чемпиона мира Владимира Крамника выступила программа Fritz. Матч из восьми игр окончился ничьей. Условия этого матча были гораздо более благоприятными для человека, и в качестве аппаратного обеспечения использовался обычный персональный компьютер, а не суперкомпьютер. Тем не менее Крамник прокомментировал этот матч так: “Теперь очевидно, что эта самая лучшая программа и чемпион мира играют примерно на равных”.

**Шашки.** Начиная с 1952 года Артур Самюэл из компании IBM в свое свободное время занимался разработкой программы игры в шашки, которая совершенствовалась с помощью обучения свою собственную функцию оценки, играя сама с собой

тысячи раз. Эта идея будет описана более подробно в главе 21. Программа Самюэла вначале играла на уровне новичка, но всего лишь через несколько дней игры с самой собой усовершенствовалась до такого уровня, что стала побеждать Самюэла (хотя он не был сильным игроком). В 1962 году эта программа победила Роберта Нили, чемпиона игры в шашки “вслепую”, благодаря ошибке с его стороны. Многие в то время посчитали, что компьютеры уже играют в шашки лучше людей, но фактически этого еще не произошло. Тем не менее, если учесть то, что вычислительное оборудование, использовавшееся Самюэлом (компьютер IBM 704), имело 10 000 слов основной памяти, магнитную ленту для долговременного хранения и процессор с частотой 0,000001 ГГц, эта победа остается большим достижением.

Превзойти данное достижение пытались многие, и, наконец, Джонатан Шеффер со своими коллегами разработал программу Chinook, которая работает на обычных персональных компьютерах и использует альфа-бета-поиск. В программе Chinook применяется заранее вычисленная база данных из всех 444 миллиардов позиций с восьмью или меньшим количеством шашек на доске, что позволяет ей играть в эндшпиле безошибочно. Программа Chinook заняла второе место в 1990 году на открытом чемпионате США и завоевала право сделать заявку на участие в мировом чемпионате. Но затем эта программа столкнулась с проблемой в лице Мэриона Тинсли. Доктор Тинсли был чемпионом мира свыше 40 лет, проиграв за все это время только три партии. В первом матче против программы Chinook Тинсли потерпел свое четвертое и пятое поражение, но выиграл матч со счетом 20,5–18,5. Матч на звание чемпиона мира в августе 1994 года между Тинсли и программой Chinook закончился преждевременно, поскольку Тинсли был вынужден сдаться из-за ухудшения состояния здоровья. Программа Chinook была официально признана чемпионом мира.

Шеффер считает, что при наличии достаточной вычислительной мощи база данных с эндшпилами может быть увеличена до такой степени, что прямой поиск из начальной позиции будет всегда достигать решенных позиций, т.е. задача игры в шашки должна быть полностью решена. (Программа Chinook иногда объявляла о своем выигрыше на пятом ходу.) Исчерпывающий анализ такого рода может быть выполнен вручную для игры в крестики-нолики 3×3 и с помощью компьютера для игр Qubic (объемные крестики-нолики 4×4×4), гомоку (пять в ряд) и Nine-Men's Morris (Мельница) [524]. В замечательной работе Кена Томпсона и Льюиса Стиллера [1464] приведены решения всех шахматных эндшпилей с пятью фигурами и некоторых эндшпилей с шестью фигурами, причем эти результаты предоставлены для всеобщего доступа в Internet. Стиллер обнаружил один вариант, в котором достигался форсированный мат, но он состоял из 262 ходов; этот результат вызвал некоторый переполох, поскольку в шахматных правилах установлено, чтобы в течение 50 ходов происходил хоть какой-то “прогресс”, иначе засчитывается ничья.

Игра  “Отелло”, называемая также “Реверси”, по-видимому, более популярна как компьютерная игра, а не настольная. Она имеет меньшее пространство поиска, чем шахматы, поскольку в ней обычно имеется от 5 до 15 допустимых ходов, но опыт в оценке позиций в ней пришлось накапливать буквально с нуля. В 1997 году программа Logistello [209] победила человека — чемпиона мира, Такеси Мураками, со счетом 6:0. Теперь общепризнано, что люди не могут соревноваться с компьютерами в игре “Отелло”.

 **Нарды.** В разделе 6.5 описано, почему из-за включения элемента неопределенности, вызванного метанием жребия, глубокий поиск становится дорогостоящим

роскошью. В области игры в нарды было предпринято много усилий по усовершенствованию функции оценки. Гэри Тесауро [1499] использовал сочетание метода обучения с подкреплением Самюэла и методов нейронных сетей (глава 20) для разработки удивительно точного средства оценки, которое использовалось при поиске на глубину 2 или 3. Сыграв против самой себя больше миллиона тренировочных игр, разработанная Гэри Тесауро программа TD-Gammon по праву заняла место среди лучших трех игроков мира. Некоторые рекомендации этой программы по выбору дебютных ходов в начальной стадии игры в нарды радикально расходились с “мудрыми” советами, передававшимися из поколения в поколение в течение многих веков.

❖ **Го** — это наиболее популярная настольная игра в Азии, которая для полного овладения мастерством требует от профессионалов столько же усилий, как шахматы. Поскольку в ней доска имеет размеры  $19 \times 19$ , коэффициент ветвления начинается с 361, а эта величина слишком обременительна для обычных методов поиска. Вплоть до 1997 года вообще не было ни одной достаточно компетентной программы, но теперь программы часто делают ходы, достойные уважения. В большинстве наилучших программ сочетаются методы распознавания шаблонов (в которых используется принцип — при появлении такого-то шаблона из камней необходимо рассмотреть такой-то ход) с ограниченным поиском (для выработки решения о том, следует ли стремиться к захвату таких-то камней, не выходя за пределы данной локальной области). Ко времени написания данной книги, по-видимому, самыми сильными программами были Goemate Чен Жишинга и Go4++ Майкла Рейса, каждая из которых достигает рейтинга, примерно равного 10 кю (уровень слабого любителя). Ведение игры го — это такая область, которая, скорее всего, достигнет развития в результате интенсивных исследований с использованием более сложных методов формирования рассуждений. Успех может быть достигнут в результате обнаружения способов интеграции нескольких линий локальных рассуждений о каждой из многих слабо связанных “субигр”, на которые может быть выполнена декомпозиция всей игры го. Подобные методы имели бы колossalную ценность для всех интеллектуальных систем в целом.

❖ **Бридж** — это игра с неполной информацией: карты любого игрока скрыты от других игроков. Кроме того, бридж — это игра с несколькими игроками, в которой участвуют четыре игрока, а не два, хотя игроки разбиты по парам на две команды. Как было показано в разделе 6.5, оптимальная игра в бридж может включать элементы сбора информации, передачи информации, введения в заблуждение и тщательного взвешивания вероятностей. Многие из этих методов используются в программе Bridge Baron™ [1441], которая выиграла в 1997 году чемпионат мира по бриджу среди компьютеров. Хотя программа Bridge Baron не играет оптимальным образом, она представляет собой одну из немногих успешно действующих систем ведения игры, в которой используются сложные, иерархические планы (см. главу 12), основанные на таких идеях высокого уровня, как **импас** (finessing) и **сквиз** (squeezing), которые знакомы игрокам в бридж.

Чемпионат мира 2000 года с большим отрывом от соперников выиграла программа GIB [559]. В программе GIB используется метод “усреднения по прогнозам” с двумя важными модификациями. Во-первых, в этой программе вместо исследования того, насколько удачным окажется каждый вариант игры по отношению к каждому варианту возможного расположения скрытых карт (количество которых может достигать 10 миллионов), исследуется случайно выбранный образец из 100 расположе-

жений. Во-вторых, в программе GIB используется **обобщение на основе объяснения** для вычисления и кэширования общих правил оптимальной игры в виде определений различных стандартных классов ситуаций. Это позволяет данной программе принимать точное решение в отношении каждой раздачи. Тактическая точность программы GIB компенсирует ее неспособность формировать рассуждения в отношении имеющейся информации. Она финишировала на 12-м месте среди 35 участников в парных соревнованиях (в которых предусматривался только розыгрыш карт, имеющихся на руках) в чемпионате мира среди людей в 1998 году, значительно превзойдя ожидания многих специалистов в этой области.

## 6.7. ОБСУЖДЕНИЕ ИЗЛОЖЕННЫХ СВЕДЕНИЙ

---

Поскольку вычисление оптимальных решений в играх чаще всего является не осуществимым, во всех алгоритмах необходимо использовать некоторые предположения и допущения. Стандартный подход, основанный на использовании минимаксных значений, функций оценки и альфа-бета-отсечения, представляет собой лишь один из способов достижения этой цели. По-видимому, из-за того, что он был предложен уже давно, этот стандартный подход интенсивно развивался и стал доминировать над другими методами, участвующими в борьбе за право на существование. Некоторые специалисты в этой области считают, что такое состояние дел стало причиной отделения проблематики ведения игр от основного направления развития исследований по искусственному интеллекту, поскольку этот стандартный подход больше не предоставляет большого простора для новых открытий, позволяющих найти ответы на общие вопросы в области принятия решений. В настоящем разделе описаны некоторые альтернативные варианты.

Вначале рассмотрим минимаксный поиск. Алгоритмы минимаксного поиска позволяют выбрать оптимальный ход в данном конкретном дереве поиска, при условии, что оценки листовых узлов являются абсолютно правильными. Но в действительности оценки обычно представляют собой грубые прогнозы стоимости той или иной позиции, поэтому можно считать, что с ними связаны существенные ошибки. На рис. 6.11 показано дерево игры с двумя полуходами, для которого минимаксный поиск представляется неподходящим. Минимаксный поиск подсказывает, что должна быть выбрана правая ветвь, тогда как весьма вероятно, что истинное значение левой ветви должно быть выше. Минимаксный выбор основывается на предположении, что все узлы, отмеченные значениями 100, 101, 102 и 100, действительно лучше, чем узел, отмеченный значением 99. Однако тот факт, что узел с отметкой 99 имеет сестринские узлы с отметками 1000, наводит на мысль, что фактически он может иметь более высокое истинное значение. Один из способов справиться с этой проблемой состоит в том, чтобы использовать какую-то оценку, которая возвращает распределение вероятностей среди возможных значений. В таком случае появляется возможность вычислить распределение вероятностей для значения родительского узла с использованием стандартных статистических методов. К сожалению, обычно значения сестринских узлов являются в высшей степени коррелированными, поэтому такое вычисление может оказаться дорогостоящим и требующим больших усилий для получения информации.

Затем рассмотрим алгоритм поиска, в котором формируется дерево. Каждый проектировщик алгоритмов стремится создать такой способ вычислений, который

действует быстро и позволяет определить хороший ход. Наиболее очевидным недостатком альфа-бета-алгоритма является то, что он предназначен не только для выбора хорошего хода, но и для расчета пределов значений всех допустимых ходов. Чтобы понять, почему эта лишняя информация часто не нужна, рассмотрим позицию, в которой имеется только один допустимый ход.

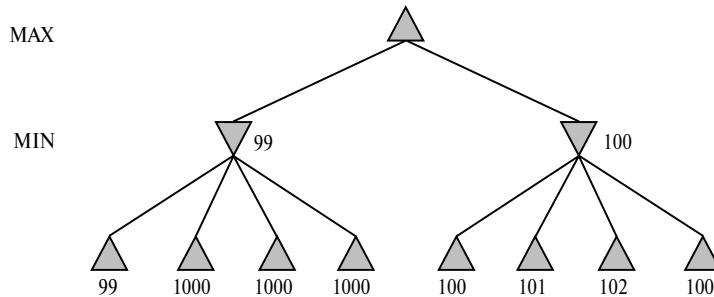


Рис. 6.11. Дерево игры с двумя полуходами, для которого минимаксный поиск может оказаться неподходящим

Альфа-бета-поиск все еще вырабатывает и оценивает большое и полностью бесполезное дерево поиска. Безусловно, можно предусмотреть в этом алгоритме какую-то проверку подобной ситуации, но она просто замаскирует основную проблему: многие вычисления, выполняемые в альфа-бета-алгоритме, практически ничего не дают для решения задачи. Ситуация, в которой имеется только один допустимый ход, немного отличается от той, где имеется несколько допустимых ходов, притом что один из них является правильным, а остальные — явно катастрофическими. В подобной ситуации наличия “четко выраженного фаворита” было бы желательно быстро достигать решения после проведения небольшого объема поиска, чем терять время, которое можно было бы использовать продуктивнее в дальнейшем, в более проблематичной позиции. Это ведет к идее полезности развертывания узла. Хороший алгоритм поиска должен выбирать варианты развертывания узлов с высокой полезностью, т.е. те варианты, которые, со всей вероятностью, приведут к обнаружению гораздо лучшего хода. Если же отсутствуют варианты развертывания узлов, полезность которых превышает их стоимость (измеряемую в затратах времени), то алгоритм должен прекратить поиск и выбрать ход. Следует отметить, что такое усовершенствование касается не только ситуаций с явными фаворитами, но и тех случаев, когда имеются симметричные ходы, для которых сколь угодно большой объем поиска не позволит показать, что один ход лучше другого.

Рассуждения о том, какие вычисления следует выполнять, а какие нет, называются **метарассуждениями** (рассуждениями о рассуждениях). Этот подход распространяется не только на ведение игр, но и в целом на рассуждения любого рода. Все вычисления выполняются для того, чтобы выработать лучшие решения, все они имеют стоимость и характеризуются определенной вероятностью достижения конкретного улучшения качества решения. Альфа-бета-поиск представляет собой реализацию метарассуждений простейшего вида, а именно теоремы о том, что некоторые ветви дерева можно игнорировать без ущерба для всей игры. Но такие метарассуждения могут проводиться гораздо лучше. В главе 16 будет показано, как сделать изложенные идеи более точными и реализуемыми.

Наконец, еще раз рассмотрим природу самого поиска. Алгоритмы эвристического поиска и ведения игр действуют путем выработки последовательностей конкретных состояний (начиная от начального состояния), а затем применения функции оценки. Безусловно, люди играют в игры иначе. В шахматах игрок часто руководствуется конкретной целью (например, поймать ферзя противника) и может использовать эту цель для избирательной выработки осуществимых планов ее достижения. Иногда такого рода подход на основе **рассуждений, управляемых целью**, или **планирования**, позволяет полностью устраниТЬ комбинаторный поиск (см. часть IV). Программа Paradise Дэвида Уилкинса [1592] — это единственная программа, в которой с успехом использовались рассуждения, управляемые целью, для игры в шахматы; она оказалась способной решать некоторые шахматные задачи, для которых требовались комбинации из 18 ходов. Тем не менее еще нет полного понимания того, как объединить эти два типа алгоритмов в надежную и эффективную систему, хотя программа Bridge Bagot может рассматриваться как шаг в правильном направлении. Полностью интегрированная система стала бы значительным достижением не только в области исследований ведения игр, но и для всех исследований по искусственному интеллекту в целом, поскольку послужила бы хорошей основой для создания интеллектуального агента общего назначения.

## РЕЗЮМЕ

---

В этой главе были проанализированы самые разные игры с тем, чтобы можно было понять, что означают слова “оптимальная игра”, а также узнать, как научиться хорошо играть на практике. Ниже изложены наиболее важные идеи, которые рассматривались в данной главе.

- Любая игра может быть определена с помощью **начального состояния** (которое показывает, как осуществляется подготовка доски к игре), допустимых **действий** в каждом состоянии, **проверки терминального состояния** (позволяющей определить, когда игра окончена) и **функции полезности**, которая применяется к терминальным состояниям.
- В играх с двумя игроками и нулевой суммой, характеризующихся **полной информацией**, для выбора оптимальных ходов с помощью перебора узлов в глубину в дереве игры может использоваться алгоритм **минимаксного поиска**.
- Алгоритм **альфа-бета-поиска** вычисляет такие же оптимальные ходы, как и алгоритм минимаксного поиска, но позволяет достичь гораздо большей эффективности, удаляя поддеревья, которые, по всей вероятности, не нужны для поиска решения.
- Обычно не представляется возможным рассматривать все дерево игры (даже с помощью альфа-бета-поиска), поэтому необходимо в какой-то точке останавливать поиск и применять **функцию оценки**, позволяющую определить приближенное значение полезности некоторого состояния.
- Ведение игр с элементами случайности можно осуществить с помощью расширения алгоритма минимаксного поиска, в котором оцениваются **узлы же-**

ребьевки путем определения средней полезности всех их дочерних узлов с учетом вероятности каждого дочернего узла.

- Для определения оптимальных ходов в играх с **неполной информацией**, таких как бридж, необходимо формировать рассуждения о текущем и будущем **доверительных состояниях** для каждого игрока. Одна из простых аппроксимаций может быть получена путем усреднения значения данного действия по всем возможным конфигурациям недостающей информации.
- Программы способны соревноваться на равных или побеждать лучших игроков-людей в шашках, игре “Отелло” и в нардах, а также вплотную приблизились к ним в бридже. Программа победила чемпиона мира по шахматам в одном показательном матче. В игре го программы до сих пор остаются на любительском уровне.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Ранняя история развития механических средств ведения игр была запятнана многочисленными случаями мошенничества. Наиболее известным из них был “Турок” барона Вольфганга фон Кемпелена (1734–1804). Это устройство выдавали за автомат для игры в шахматы и смогли с его помощью обмануть очень многих, в том числе Наполеона. Лишь в дальнейшем выяснилось, что это устройство представляет собой хитроумный ящик фокусника, в котором сидит человек, хорошо играющий в шахматы [919]. Игры с участием этого устройства проводились с 1769 по 1854 годы. В 1846 году Чарльз Бэббидж (который был в восторге от “Турка”) участвовал в одной из первых серьезных дискуссий на тему осуществимости задачи ведения игр в шахматы и шашки с помощью вычислительного устройства [1089]. Он также спроектировал, но не построил машину специального назначения для игры в крестики-нолики. Первая настоящая машина для ведения игры была построена примерно в 1890 году испанским инженером Леонардо Торресом и Кеведо. Она была специально предназначена для ведения шахматного эндшпилля “KRK” (King and Rook vs. King — король и ладья против короля) и гарантировала победу из любой позиции в игре с этими тремя фигурами.

В качестве первоисточника идеи алгоритма минимаксного поиска часто называют статью, опубликованную в 1912 году Эрнстом Цермело, основоположником современной теории множеств [1642]. К сожалению, эта статья содержала несколько ошибок, а алгоритм минимаксного поиска не был в ней описан правильно. Солидный фундамент теории игр был создан в оригинальной работе *Theory of Games and Economic Behavior* [1546], которая содержала анализ, показывающий, что для некоторых игр требуются стратегии, которые являются рандомизированными (или становятся непредсказуемыми для противников по каким-то иным причинам). Дополнительная информация на эту тему приведена в главе 17.

На заре компьютерной эры возможность создания компьютерных шахмат интересовала многих важных деятелей в этой области. Конрад Цузе [1651] (первый, кто спроектировал программируемый компьютер) разработал довольно подробные предложения, касающиеся того, как может быть решена эта задача. Во влиятельной книге *Cybernetics* (*Кибернетика*) Норberta Винера [1589] обсуждался один возможный

проект создания шахматной программы и были высказаны идеи минимаксного поиска, останова на заданной глубине и применения функций оценки. Клод Шенон [1395] изложил основные принципы создания современных программ ведения игр гораздо более подробно, чем Винер. Он ввел понятие поиска спокойной позиции, а также высказал некоторые идеи в отношении избирательного (неисчерпывающего) поиска в дереве игры. Слейтор [1430] и комментаторы его статьи также рассматривали возможности ведения игры в шахматы с помощью компьютера. В частности, Гуд [574] разработал понятие спокойной позиции независимо от Шенона.

В 1951 году Алан Тьюринг написал первую компьютерную программу, способную вести полную игру в шахматы [1521]. Но программа Тьюринга фактически никогда не эксплуатировалась на компьютере; она была испытана путем моделирования вручную в партии против очень слабого игрока-человека, который ее победил. Между тем, Д.Г. Принц [1238] написал и проверил на практике программу, которая решала шахматные задачи, но не могла вести полную игру. Первая программа<sup>3</sup>, способная вести полную игру в стандартные шахматы, была написана Алексом Бернштейном [112], [113].

Джон Маккарти сформулировал идею альфа-бета-поиска в 1956 году, но не опубликовал свое открытие. В шахматной программе NSS [1128] используется упрощенная версия альфа-бета-поиска; она представляет собой первую шахматную программу, в которой было введено это усовершенствование. По данным Нильссона [1141], в программе игры в шашки Артура Самюэла [1349], [1350] также использовался альфа-бета- поиск, хотя Самюэл об этом не упоминал в опубликованных отчетах о своей системе. Статьи с описанием альфа-бета-поиска были опубликованы в начале 1960-х годов [198], [625], [1427]. Одна из реализаций полного альфа-бета-поиска описана Слаглем и Диксоном [1428] применительно к программе ведения игры калах. Кроме того, альфа-бета- поиск использовался в шахматной программе “Kotok-McCarthy”, написанной студентом Джоном Маккарти [847]. Кнут и Мур [810] изложили историю создания алгоритма альфа-бета-поиска, а также привели доказательство его правильности и представили результаты анализа временной сложности. Проведенный ими анализ альфа-бета-поиска со случайным упорядочиванием преемников показал его асимптотическую сложность  $O((b/\log b)^d)$ , что представляло собой довольно мрачный прогноз, поскольку соответствующий ему эффективный коэффициент ветвления  $b/\log b$  не намного меньше, чем само значение  $b$ . После этого они отметили, что указанная асимптотическая формула является точной только для  $b > 1000$  или таких порядков величин, тогда как часто упоминаемое значение  $O(b^{3d/4})$  относится к тому диапазону коэффициентов ветвления, который чаще всего встречается в настоящих играх. Перл [1187] показал, что альфа-бета- поиск является асимптотически оптимальным среди всех алгоритмов поиска в дереве игры на постоянную глубину.

В первом шахматном матче между компьютерами участвовала программа Kotok-McCarthy и программа “ITEP” (Institute of Theoretical and Experimental Physics), написанная в середине 1960-х годов в Московском институте теоретической и экспериментальной физики (ИТЕФ) [4]. Этот межконтинентальный матч проводился по телеграфу. Он закончился в 1967 году победой программы ITEP со счетом 3:1. Первой шахматной программой, которая успешно соревновалась с людьми, была про-

<sup>3</sup> В [1128] упоминается программа, написанная в Советском Союзе для компьютера БЭСМ, которая могла предшествовать программе Бернштейна.

грамма MacHack 6 [594]. Ее рейтинг, примерно равный 1400, был намного выше 1000 — уровня начинающего, но все же далеко не достигал рейтинга 2800 или больше, который требовался для выполнения предсказания Герберта Саймона, высказанного в 1957 году, что компьютерная программа станет чемпионом мира по шахматам через 10 лет [1419].

Начиная с первого Североамериканского чемпионата ACM по компьютерным шахматам, проведенного в 1970 году, между шахматными программами разгорелась серьезная конкуренция. Программы, разработанные в начале 1970-х годов, стали чрезвычайно сложными, насыщенными всевозможными ухищрениями, которые были предназначены для устранения некоторых ветвей поиска, выработки приемлемых ходов и т.д. В 1974 году в Стокгольме проводился первый чемпионат мира по компьютерным шахматам, в котором победила Kaissa [5], еще одна программа, разработанная в ИТЕФ. В программе Kaissa использовался гораздо более прямолинейный подход к организации исчерпывающего алфа-бета-поиска в сочетании с поиском спокойных позиций. Перспективность этого подхода была подтверждена убедительной победой программы Chess 4.6 на чемпионате мира по компьютерным шахматам в 1977 году. Программа Chess 4.6 исследовала до 400 000 позиций за каждый ход и имела рейтинг 1900.

Последняя версия разработанной Гринблаттом программы MacHack 6 была первой шахматной программой, которая эксплуатировалась на специализированном аппаратном обеспечении, разработанном специально для шахмат [1094], но первой программой, в которой удалось добиться заметного успеха благодаря использованию заказного аппаратного обеспечения, была программа Belle [286]. Применявшиеся в программе Belle аппаратное обеспечение для выработки ходов и оценки позиции позволяло ей исследовать несколько миллионов позиций за каждый ход. Программа Belle достигла рейтинга 2250 и стала первой программой уровня мастера. В университете CMU бывшим чемпионом мира по игре в шахматы по переписке Хансом Берлинером и его студентом Карлом Эбелингом была разработана система Hitech, также представлявшая собой компьютер специального назначения, который обеспечивал быстрое вычисление функций оценки [428], [108]. Система Hitech, которая вырабатывала около 10 миллионов позиций за ход, стала чемпионом Северной Америки по компьютерным шахматам в 1985 году и оказалась первой программой, которая смогла победить гроссмейстера-человека в 1987 году. Система Deep Thought, которая также была разработана в университете CMU, явилась еще одним шагом в направлении чистого ускорения поиска [697]. Она достигла рейтинга 2551 и стала предшественником системы Deep Blue. В 1980 году была основана премия Фредкина (Fredkin), в которой было предложено 5000 долларов за первую программу, достигшую уровня мастера, 10 000 долларов за первую программу, достигшую рейтинга 2500 USCF (United States Chess Federation — Шахматная федерация США) (что примерно соответствует уровню гроссмейстера), и 100 000 долларов за первую программу, победившую чемпиона мира по шахматам. Премию 5000 долларов получила программа Belle в 1983 году, премию 10 000 долларов — система Deep Thought в 1989 году, а премию 100 000 долларов — система Deep Blue за победу над Гарри Каспаровым в 1997 году. Необходимо учитывать, что успех Deep Blue был обусловлен не только усовершенствованием аппаратных средств, но и применением лучших алгоритмов [216], [696]. Применение таких методов, как эвристика нулевого хода [90], привело к созданию программ, которые стали весьма избирательными в своих поисках. В последних трех чемпионатах мира по компьютерным шахматам, проводив-

шихся в 1992, 1995 и 1999 годах, победили программы, работающие на стандартных персональных компьютерах. По-видимому, наиболее полное описание современной шахматной программы предоставлено Эрнстом Хейнцем [644], чья программа DarkThought получила самый высокий рейтинг среди некоммерческих программ для персональных компьютеров на чемпионате мира 1999 года.

Для преодоления проблем, связанных со “стандартным подходом”, кратко описанных в разделе 6.7, было предпринято несколько попыток. По-видимому, первым избирательным алгоритмом поиска, в котором учитывались некоторые теоретические обоснования методов сокращения поиска, был В\* [105], в котором предпринята попытка устанавливать интервальные пределы возможных значений узла в дереве игры, а не давать единственную точечную оценку. Листовые узлы выбираются для развертывания в целях уточнения пределов верхнего уровня до тех пор, пока не обнаруживается ход, который “безусловно является наилучшим”. Палей [1165] развил идею алгоритма В\*, используя распределения вероятностей значений вместо интервалов. В алгоритме поиска конспиративного числа (conspiracy number search) Дэвида Маккаллестера [1004] предусмотрено развертывание листовых узлов, которые, изменения свои значения, могут вынудить программу предпочесть новый ход от корня. В алгоритме MGSS\* [1331] используются описанные в главе 16 методы теории решений для оценки стоимости развертывания каждого листа с точки зрения ожидаемого усовершенствования качества решения, формируемого от корня. Этот алгоритм превзошел альфа-бета-алгоритм, применяемый в игре “Отелло”, несмотря на то, что в нем проводился поиск среди узлов, количество которых было на порядок меньше. Вообще говоря, подход, принятый в алгоритме MGSS\*, может применяться для управления размышлениями в любой форме.

Альфа-бета-поиск во многих отношениях представляет собой аналог поиска на основе метода ветвей и границ, который был превзойден поиском А\* в случае с одним агентом, если не считать того, что альфа-бета-поиск рассчитан на двух игроков. Алгоритм SSS\* [1466] может рассматриваться как поиск А\* для двух игроков; в нем для достижения того же решения никогда не развертывается больше узлов, чем в алгоритме альфа-бета-поиска. В его первоначальной форме алгоритм SSS\* предъявлял чрезмерные требования к памяти и характеризовался значительными вычислительными издержками на сопровождение очереди, поэтому был практически не применимым. Тем не менее на основе алгоритма RBFS была разработана версия SSS\* с линейно возрастающими потребностями в пространстве [843]. Плат и др. [1214] разработали новый подход к реализации алгоритма SSS\* как комбинации альфа-бета-поиска и таблиц транспозиции, показали, как преодолеть недостатки первоначального алгоритма, и создали новый вариант, называемый MTD(f), который нашел свое применение во многих превосходных программах.

Д.Ф. Бил [89] и Дана Най [1113], [1114] изучали недостатки минимаксного поиска, касающиеся приближенных оценок. Они показали, что при некоторых предположениях о независимости распределения листовых значений в дереве применение минимаксного поиска может привести к получению таких значений для корня, которые фактически являются менее надежными, чем полученные с помощью непосредственного использования самой функции оценки. В книге Перла *Heuristics* [1188] дано частичное объяснение этого кажущегося парадокса и приведен анализ многих алгоритмов ведения игр. Баум и Смит [83] предложили заменить алгоритм минимаксного поиска алгоритмом, основанном на учете вероятностей, и показали, что он приводит к осуществлению лучших вариантов выбора в некоторых играх.

Но объем теоретических результатов исследования влияния останова поиска на разных уровнях и применения функций оценки все еще остается недостаточным.

Алгоритм поиска на основе ожидаемого минимаксного значения был предложен Дональдом Мичи [1043], хотя, безусловно, он непосредственно следует из принципов оценки дерева игры, выдвинутых фон Нейманом и Моргенштерном. Брюс Боллард [65] распространил метод альфа-бета-отсечения на деревья с узлами жеребьевки. Первый успешно действующий программой игры в нарды была разработанная Берлиннером программа BKG [104], [107]; в ней использовалась сложная, сконструированная вручную функция оценки, а поиск осуществлялся только на глубину 1. Это была первая программа, которая смогла победить чемпиона мира — человека в одной из основных классических игр [106]. По завершении соревнований Берлиннер сразу же объявил, что это был очень короткий показательный матч (а не матч чемпионата мира) и что программе BKG очень повезло со жребием. Работа Герри Тесауро, приведшая вначале к созданию программы Neurogammon [1498], а затем TD-Gammon [1500], показала, что гораздо лучшие результаты могут быть достигнуты с помощью обучения с подкреплением. Эта тема будет рассматриваться более подробно в главе 21.

Первой из классических игр, полное решение которой было найдено компьютером, оказались не шахматы, а шашки. Кристофер Стрейчи [1469] написал первую работоспособную программу игры в шашки. Шеффер [1357] составил очень доступный отчет о разработке программы Chinook, ставшей чемпионом мира по шашкам, в котором все факты изложены “без прикрас”.

Первые программы для игры го были разработаны немного позднее по сравнению с программами для шашек и шахмат [906], [1281], а их развитие происходило более медленно. Райдер [1334] использовал подход, основанный исключительно на использовании поиска в сочетании с всевозможными методами избирательного отсечения для преодоления колossalного коэффициента ветвления, характерного для этой игры. Зобрист [1650] применил правила “условие–действие”, которые способны предложить приемлемые ходы при обнаружении известных шаблонов. Рейтман и Уилкокс [1280] добились хороших результатов, сочетая применение правил и поиска, поэтому большинство современных программ было разработано на основе этого гибридного подхода. Мюллер [1103] подытожил современное состояние работы в области компьютеризации игры го и привел в своей книге много ссылок. Аншлевич [33] использовал подобные методы для создания программы для игры гекс. Описания современных разработок можно найти в журнале *Computer Go Newsletter*, который публикует организация Computer Go Association.

Работы по компьютерному ведению игры появляются во многих разных источниках. В трудах конференции *Heuristic Programming in Artificial Intelligence*, имеющей довольно дезориентирующее название, публикуются отчеты о компьютерных олимпиадах, в рамках которых проводятся весьма разнообразные игры. Имеется также несколько отредактированных сборников важных статей об исследованиях в области ведения игр [920], [921], [988]. Международная ассоциация компьютерных шахмат (International Computer Chess Association — ICCA), основанная в 1977 году, публикует ежеквартальный журнал *ICGA Journal* (который раньше носил название *ICCA Journal*). В сериино выпускаемой антологии *Advances in Computer Chess* публикуются важные статьи, начиная со статьи Кларке [268]. В томе 134 журнала *Artificial Intelligence* (2002) содержатся описания современных программ для шахмат, “Отелло”, гекса, сёги (японские шахматы), го, нард, покера, Scrabble™ (“Эрудит”) и др.

## УПРАЖНЕНИЯ

- 6.1.** В этой задаче рассматриваются основные концепции ведения игр с использованием в качестве примера игры в крестики-нолики. Определим  $X_n$  как количество строк, столбцов или диагоналей, в которых находится точно  $n$  значков X и ни одного значка O. Аналогичным образом,  $O_n$  — это количество строк, столбцов или диагоналей, в которых находятся только  $n$  значков O. Функция полезности присваивает +1 любой позиции с  $X_3=1$  и -1 любой позиции с  $O_3=1$ . Все остальные терминальные позиции имеют полезность 0. Для нетерминальных позиций используется линейная функция оценки, определенная как  $\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$ .
- a) Сколько приблизительно существует возможных вариантов игры в крестики-нолики?
  - b) Покажите полное дерево игры, начиная от пустой доски вплоть до глубины 2 (т.е. глубины, на которой на доске имеется один значок X и один значок O), учитывая симметрию позиций.
  - c) Проставьте в этом дереве оценки всех позиций на глубине 2.
  - d) С использованием алгоритма минимаксного поиска проставьте в дереве зарезервированные значения для позиций на глубине 1 и 0 и воспользуйтесь этими значениями для выбора наилучшего начального хода.
  - d) Обведите кружками узлы на глубине 2, которые не оценивались бы в случае применения альфа-бета-отсечения, исходя из предположения, что эти узлы вырабатываются в порядке, оптимальном для альфа-бета-отсечения.
- 6.2.** Докажите следующее утверждение: для каждого дерева игры значение полезности, полученное игроком MAX с использованием минимаксных решений в игре против неоптимально действующего игрока MIN, никогда не будет ниже значений полезности, полученных в игре против оптимально действующего игрока MIN. Можете ли вы показать дерево игры, в котором игрок MAX может действовать еще лучше, используя неоптимальную стратегию против неоптимально действующего игрока MIN?
- 6.3.** Рассмотрим игру с двумя игроками, показанную на рис. 6.12.



*Рис. 6.12. Начальная позиция простой игры. Игрок А ходит первым. Два игрока ходят по очереди, и каждый игрок должен переместить свою фишку в открытый смежный квадрат в любом направлении. Если противник занимает смежный квадрат, то игрок может перенести свою фишку через фишку противника на следующий открытый квадрат, если он имеется. (Например, если фишка А стоит в квадрате 3, а фишка В — в квадрате 2, то А снова может перейти в квадрат 1.) Игра заканчивается после того, как один из игроков достигает противоположного конца доски. Если игрок А достигает квадрата 4 первым, то значение этой игры для А равно +1; если игрок В достигает квадрата 1 первым, то значение этой игры для игрока А равно -1.*

- a) Нарисуйте полное дерево игры с использованием следующих соглашений:

- записывайте каждое состояние как  $(s_A, s_B)$ , где  $s_A$  и  $s_B$  обозначают местонахождения фишек;
  - обводите квадратом каждое терминальное состояние и записывайте для него значение игры в кружке;
  - обводите двойными квадратами циклические состояния (состояния, которые уже появлялись на пути к корню). Поскольку не ясно, как присваивать значения циклическим состояниям, обозначайте каждое из них знаком ? в кружке.
- 6) Обозначьте каждый узел его зарезервированным минимаксным значением (также в кружке). Объясните, как вы учитывали значения ? и почему.
- в) Объясните, почему стандартный алгоритм минимаксного поиска в этом дереве игры потерпит неудачу, и вкратце опишите, каким образом вы могли бы его исправить, опираясь на ваш ответ на пункт б). Приведет ли модифицированный вами алгоритм к получению оптимальных решений для всех игр с циклами?
- г) Эту игру с четырьмя клетками можно обобщить до  $n$  клеток для любого  $n > 2$ . Докажите, что игрок А побеждает, если число  $n$  — четное, и проигрывает, если число  $n$  — нечетное.
- 6.4.  Реализуйте генераторы хода и функции оценки для одной или нескольких из следующих игр: калах, “Отелло”, шашки и шахматы. Сконструируйте агента общего назначения для ведения игры на основе альфа-бета-поиска, в котором используется ваша реализация. Сравните эффективность увеличения глубины поиска, усовершенствования способа упорядочения ходов и улучшения функции оценки. Насколько близко приближается полученный вами эффективный коэффициент ветвления к идеальному случаю самого совершенного упорядочения ходов?
- 6.5. Разработайте формальное доказательство правильности алгоритма альфа-бета-отсечения. Для этого рассмотрите ситуацию, показанную на рис. 6.13. Вопрос заключается в том, следует ли выполнять отсечение в узле  $n_j$ , который представляет собой узел MAX и является одним из потомков узла  $n_1$ . Основная идея такова, что отсечение должно выполняться тогда и только тогда, когда можно показать, что минимаксное значение  $n_1$  не зависит от значения  $n_j$ .
- а) Значение узла  $n_1$  определяется с помощью выражения
- $$n_1 = \min(n_2, n_{21}, \dots, n_{2b_2})$$
- Найдите аналогичное выражение для узла  $n_2$ , а следовательно, выражение для узла  $n_1$  в терминах значений узла  $n_j$ .
- б) Допустим, что  $l_i$  — минимальное (или максимальное) значение узлов слева от узла  $n_i$  на глубине  $i$ , минимаксное значение которого уже известно. Аналогичным образом, допустим, что  $r_i$  — минимальное (или максимальное) значение неисследованных узлов справа от  $n_i$  на глубине  $i$ . Перепишите ваше выражение для узла  $n_1$  в терминах значений  $l_i$  и  $r_i$ .
- в) Теперь переформулируйте это выражение для того, чтобы показать, что значение узла  $n_j$  не должно превышать некоторый предел, полученный на основе значений  $l_i$ , для того, чтобы оно могло повлиять на значение узла  $n_1$ .

- г) Повторите этот процесс для того случая, когда  $n_j$  является узлом MIN.

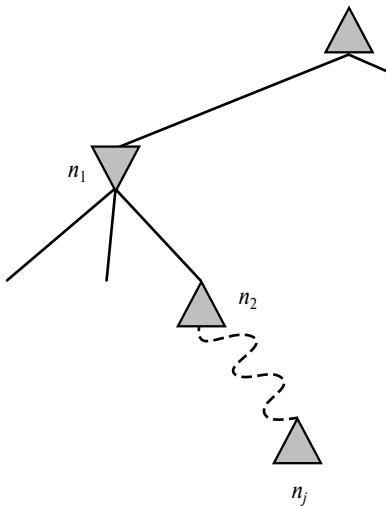


Рис. 6.13. Ситуация, возникающая при определении того, следует ли отсекать узел  $n_j$

- 6.6. Реализуйте алгоритм поиска по ожидаемому минимаксному значению и алгоритм \*-альфа-бета-поиска, который описан Баллардом [65], для отсечения деревьев игры с узлами жеребьевки. Испытайте эти алгоритмы в такой игре, как нарды, и проведите измерения эффективности отсечения в \*-альфа-бета-поиске.
- 6.7. Докажите, что после положительной линейной трансформации листовых значений (т.е. трансформации значения  $x$  в  $ax+b$ , где  $a>0$ ) выбор хода в дереве игры остается неизменным, даже если в нем имеются узлы жеребьевки.
- 6.8. Рассмотрите описанную ниже процедуру выбора ходов в играх с узлами жеребьевки:
- Сформируйте некоторые последовательности метания жребия (скажем, 50) вплоть до подходящей глубины (скажем, 8).
  - После того как результаты метания жребия становятся известными, дерево игры преобразуется в детерминированное. Для каждой последовательности метания жребия найдите решение результирующего детерминированного дерева игры с помощью альфа-бета-поиска.
  - С использованием этих результатов оцените значение каждого хода и выберите наилучший.
- Будет ли эта процедура действовать успешно? Почему да (или почему нет)?
- 6.9. Опишите и реализуйте среду ведения игры в реальном времени с несколькими игроками, где время образует часть состояния среды, а игрокам выделяются фиксированные промежутки времени.
- 6.10. Опишите или реализуйте описания состояний, генераторы ходов, проверки терминальных состояний, функции полезности и функции оценки для одной или нескольких из следующих игр: монополия, Scrabble (“Эрудит”), бридж

(под этим подразумевается вариант бриджа с заключением соглашения) и покер (выберите свой любимый вариант).

- 6.11.** Тщательно изучите, как взаимодействуют события жеребьевки и частичная информация в каждой из игр, упомянутых в упр. 6.10.
- Для какой из них подходит стандартная модель на основе ожидаемых минимаксных значений? Реализуйте этот алгоритм и примените его в вашем агенте для ведения игры после внесения соответствующих изменений в среду ведения игры.
  - Для какой из этих игр подходит схема, описанная в упр. 6.8?
  - Обсудите вопрос о том, как можно было бы воспользоваться фактом, что в некоторых играх игроки не имеют одинаковые знания о текущем состоянии.
- 6.12.** В алгоритме минимаксного поиска принято предположение, что игроки делают ходы по очереди, но в таких карточных играх, как вист и бридж, следующую взятку разыгрывает первым тот, кто взял предыдущую взятку.
- Модифицируйте алгоритм таким образом, чтобы он правильно действовал в этих играх. Для этого можно принять предположение, что имеется функция  $\text{Winner}(s)$ , которая сообщает, какой игрок взял только что разыгранную взятку (если она имеется).
  - Нарисуйте дерево игры для первой пары карт, имеющихся на руках, показанной на с. 262.
- 6.13.** В программе игры в шашки Chinook широко используются базы данных с эндишилями, которые предоставляют точные значения для каждой позиции с восемью или меньшим количеством шашек. Каким образом можно обеспечить эффективное формирование таких баз данных?
- 6.14.** Обсудите вопрос о том, насколько успешно может применяться стандартный подход к ведению игры в таких играх, как теннис, пул (род игры на бильярде) и крокет, которые происходят в непрерывном физическом пространстве состояний.
- 6.15.** Опишите, как изменятся алгоритмы минимаксного поиска и альфа-бета-поиска для случая игр с **ненулевой суммой** с двумя игроками, в которых каждый игрок имеет собственную функцию полезности. Может быть принято предположение, что каждый игрок знает функцию полезности другого игрока. Если в этих двух функциях нет ограничений полезности терминальных состояний, возможно ли отсечение какого-либо узла с помощью альфа-бета-поиска?
- 6.16.** Предположим, что имеется шахматная программа, способная оценивать 1 миллион узлов в секунду. Выберите компактное представление состояния игры для хранения в таблице транспозиций. Сколько примерно записей удастся вам поместить в таблицу на 500 Мбайт, находящуюся в памяти? Будет ли этого достаточно для поиска в течение трех минут, отведенных на один ход? Сколько операций поиска в таблице вы сможете выполнить за время, которое потребуется для одной оценки? Теперь предположим, что таблица транспозиций больше, чем для нее отведено оперативной памяти. Сколько узлов вы сможете оценить за время, которое требуется для выполнения одного поиска на диске с использованием стандартного аппаратного обеспечения жесткого диска?



## Часть III

### ЗНАНИЯ И РАССУЖДЕНИЯ

Логические агенты	282
Логика первого порядка	341
Логический вывод в логике первого порядка	380
Представление знаний	440

## 7 ЛОГИЧЕСКИЕ АГЕНТЫ

*В данной главе описано, как проектировать агентов, которые обладают способностью формировать представления о мире, используют процесс логического вывода для получения новых представлений о мире, а также применяют эти новые представления для определения того, что следует делать.*

В этой главе приведено вводное описание агентов, действующих на основе знаний (или просто *агентов на основе знаний*). Рассматриваемые здесь понятия (представление знаний и процессы рассуждения, которые связывают знания с действительностью) являются центральными во всей сфере искусственного интеллекта.

Вполне очевидно, что люди многое знают и обладают способностью рассуждать. Кроме того, знания и рассуждения очень важны для искусственных агентов, поскольку обеспечивают формирование успешных способов поведения, которых было бы очень трудно добиться иным образом. В этой главе будет показано, что знания о результатах действий позволяют агентам, решающим задачи, успешно действовать в сложных вариантах среды. В отличие от них рефлексные агенты были способны найти путь от Арада до Бухареста только благодаря слепой удаче. Однако знания агентов, решающих задачи, являются очень специфичными и недостаточно гибкими. Шахматная программа способна рассчитать допустимые ходы для короля того цвета, за который она играет, но не обладает многими другими полезными сведениями, например, о том, что ни одна фигура не может стоять одновременно на двух разных клетках. Агенты, основанные на знаниях, способны воспользоваться знаниями, выраженными в очень общих формах, комбинируя и рекомбинируя информацию в соответствии с бесчисленным множеством внешних условий. Часто этот процесс может быть весьма далеким от потребностей текущего момента; это можно сравнить с тем, как математик доказывает абстрактную теорему или астроном вычисляет ожидаемую продолжительность существования Земли.

Кроме того, знания и рассуждения играют решающую роль, когда приходится действовать в частично наблюдаемых вариантах среды. Агент, основанный на знаниях, способен сочетать общие знания с результатами текущих восприятий, чтобы иметь возможность выявить скрытые аспекты текущего состояния, прежде чем выбирать действия. Например, терапевт диагностирует пациента (т.е. выявляет болезненное состояние, которое недоступно непосредственному наблюдению), прежде чем выбрать способ лечения. Некоторая часть знаний, используемых терапевтом,

находится в форме правил, полученных с помощью учебников и учителей, а еще одна часть представлена в форме ассоциативных образов, которые терапевт не всегда может описать словами. Но если эти ассоциации складываются в уме терапевта, они также относятся к области знаний.

Для понимания естественного языка требуется также выявление скрытых аспектов состояния, в частности намерений говорящего. Услышав фразу: “Джон увидел алмаз через окно и страстно пожелал его получить”, мы знаем, что слово “его” относится к алмазу, а не к окну; мы рассуждаем, возможно даже подсознательно, с помощью имеющихся у нас знаний об относительной ценности этих предметов. Аналогичным образом, услышав фразу: “Джон бросил кирпич в окно и разбил его”, мы понимаем, что слово “его” относится к окну. Рассуждения позволяют нам справиться практически с бесконечным количеством форм выражения мысли, используя конечный запас обыденных знаний. Сталкиваясь с неоднозначностью подобного рода, агенты, решающие задачи, испытывают затруднения, поскольку применяемый в них способ представления задач с непредвиденными ситуациями обуславливает экспоненциальный рост количества рассматриваемых вариантов.

Не менее важная причина, по которой следует заниматься изучением агентов, основанных на знаниях, состоит в том, что такие агенты характеризуются значительной гибкостью. Они способны принимать к исполнению новые задачи, выраженные в форме явно поставленных целей, они могут быстро достигать компетентности, получая инструкции или усваивая новые знания, полученные из своей среды, кроме того, они способны приспосабливаться к изменениям в своей среде, обновляя соответствующие знания.

Изложение темы, рассматриваемой в данной главе, начинается с описания общего проекта агента, приведенного в разделе 7.1. В разделе 7.2 представлена новая простая среда (мир вымышленных существ — вампиров) и показано, как функционирует агент, основанный на знаниях, без углубления в какие-либо технические подробности. После этого в разделе 7.3 представлены основные принципы **логики**. Логика будет служить основным средством представления знаний во всей части III данной книги. Знания логических агентов всегда являются определенными — каждое высказывание в этом мире является либо истинным, либо ложным, хотя агент может не знать о существовании некоторых высказываний.

С точки зрения изложения учебного материала логика обладает тем преимуществом, что служит простым примером одного из способов представления для агентов, основанных на знаниях, но логика имеет некоторые серьезные ограничения. Очевидно, что значительная часть рассуждений, осуществляемых людьми и другими агентами в частично наблюдаемых вариантах среды, основана на оперировании знаниями, которые являются неопределенными. Логика не позволяет должным образом представить такую неопределенность, поэтому в части V рассматривается теория вероятностей, которая дает такую возможность. В частях VI и VII рассматриваются многие другие способы представления (в том числе основанные на континуальной математике), такие как сочетания гауссовых распределений, нейронных сетей и других представлений.

В разделе 7.4 этой главы представлена простая логика, называемая **пропозициональной логикой**. Несмотря на то что она является гораздо менее выразительной по сравнению с **логикой первого порядка** (глава 8), пропозициональная логика может служить для иллюстрации всех основных понятий логики. Кроме того, существует

хорошо разработанная технология формирования рассуждений в пропозициональной логике, которая будет описана в разделах 7.5 и 7.6. Наконец, в разделе 7.7 рассматривается сочетание понятия логических агентов с технологией пропозициональной логики, применяемое для создания некоторых простых агентов, действующих в мире вампуса. В этом разделе указаны некоторые ограничения пропозициональной логики, которые послужили стимулом к разработке более мощных версий логики, рассматриваемых в следующих главах.

## 7.1. АГЕНТЫ, ОСНОВАННЫЕ НА ЗНАНИЯХ

Центральным компонентом любого агента, основанного на знаниях, является его **база знаний**, или сокращенно KB (Knowledge Base). Неформально базу знаний можно определить как множество **высказываний**. (Здесь слово “высказывание” используется в качестве формального термина. Смыл этого термина близок, но не идентичен понятию высказывания в английском и других естественных языках.) Каждое высказывание выражено на языке, называемом **языком представления знаний**, и представляет некоторое утверждение о мире.

Должен существовать определенный способ добавления новых высказываний к базе знаний, а также способ извлечения из этой базы тех знаний, которые в ней содержатся. Стандартными названиями для этих операций являются соответственно Tell и Ask. Обе такие операции могут быть связаны с проведением **логического вывода**, т.е. могут потребовать получения новых высказываний из старых. В **логических агентах**, которые служат основной темой исследования данной главы, логический вывод должен подчиняться тому фундаментальному требованию, что ответ на запрос к базе знаний, переданный с помощью операции Ask, должен следовать из того, что было сообщено базе знаний (или точнее, введено с помощью операции Tell) до сих пор. Ниже будет дано более точное определение важного понятия “логического следствия”. А на данный момент будем считать, что в соответствии с этим понятием в процессе логического вывода не должны выполняться операции, не подчиняющиеся строгим правилам.

Общая схема программы агента, основанного на знаниях, приведена в листинге 7.1. Как и все агенты, описанные в данной книге, этот агент принимает на входе результаты акта восприятия *percept* и возвращает действие *action*. Агент поддерживает базу знаний, *KB*, которая может первоначально содержать некоторые **фоновые знания**. После каждого вызова программы агента выполняет три этапа. Во-первых, программа вводит в базу знаний с помощью операции Tell результаты акта восприятия, во-вторых, передает в базу знаний с помощью операции Ask запрос о том, какое действие следует предпринять. В процессе поиска ответа на этот запрос могут быть проведены исчерпывающие рассуждения в отношении текущего состояния мира, результатов возможных последовательностей действий и т.д. В-третьих, агент регистрирует свой выбор с помощью операции Tell и выполняет действие. Вторая операция Tell необходима для передачи в базу знаний информации о том, что гипотетическое действие *action* действительно было выполнено.

Подробные сведения о языке представления скрыты в трех функциях, которые реализуют интерфейс между датчиками и исполнительными механизмами, а также

между основным представлением и системой формирования рассуждений. Функция `Make-Percept-Sentence` формирует высказывание, подтверждающее, что агент получил результаты данного конкретного акта в текущий момент времени. Функция `Make-Action-Query` формирует высказывание, представляющее собой запрос о том, какое действие должно быть выполнено в текущий момент времени. Наконец, функция `Make-Action-Sentence` формирует высказывание, подтверждающее, что выбранное действие было выполнено. Подробные сведения о механизме логического вывода скрыты внутри функций `Tell` и `Ask`. Эти подробные сведения будут представлены в следующих разделах.

### Листинг 7.1. Универсальный агент, основанный на знаниях

---

```
function KB-Agent(percept) returns действие action
    static: KB, база знаний
            t, счетчик, обозначающий время, первоначально равный 0
    Tell(KB, Make-Percept-Sentence(percept, t))
    action  $\leftarrow$  Ask(KB, Make-Action-Query(t))
    Tell(KB, Make-Action-Sentence(action, t))
    t  $\leftarrow$  t + 1
    return action
```

---

Алгоритм агента, приведенный в листинге 7.1, внешне кажется весьма аналогичным алгоритмам агентов с поддержкой внутреннего состояния, описанных в главе 2. Однако в силу приведенных выше определений операций `Tell` и `Ask` агента, основанного на знаниях, уже нельзя рассматривать как произвольную программу для вычисления действий. Он уже больше подходит для описания на **уровне знаний**, в котором для фиксации его поведения требуется указать лишь то, что известно агенту и каковы его цели. Например, автоматизированный агент — водитель такси может иметь своей целью доставку пассажира в район Марин Каунти и знать, что этот район находится в г. Сан-Франциско и что мост “Золотые ворота” является единственным возможным связующим звеном между пунктом отправления и пунктом назначения. В таком случае вполне можно рассчитывать на то, что автоматизированный водитель поедет через мост “Золотые ворота”, поскольку ему известно, что именно так он достигнет цели. Следует отметить, что данный анализ не зависит от того, как будет действовать водитель такси на **уровне реализации**. Не имеет никакого значения, реализованы ли его знания географии в виде связных списков или растровых изображений карт, а также проводит ли он рассуждения, манипулируя строками символов, хранящимися в регистрах, или распространяя зашумленные сигналы по сетям из нейронов.

Как уже упоминалось во введении к этой главе, **агент, основанный на знаниях, может быть создан путем передачи ему с помощью операции Tell той информации, которую ему требуется знать**. Первоначальная программа агента, применяемая до того, как агент начнет получать результаты актов восприятия, создается путем добавления одного за другим тех высказываний, которые представляют знания проектировщика о данной среде. А если удается спроектировать язык представления, позволяющий легко выражать эти знания в форме высказываний, то задача создания агента чрезвычайно упрощается. Описанный выше подход называется **декларативным**.

подходом к созданию системы. В отличие от этого **процедурный** подход требует представления желаемых правил поведения непосредственно в виде кода программы. Но сведение к минимуму роли явных форм представления и формирования рассуждений позволяет добиться создания гораздо более эффективных систем. Агенты обоих этих типов рассматриваются в разделе 7.7. В 1970–1980-х годах приверженцы этих двух подходов вступили в ожесточенные дебаты. Однако теперь стало ясно, что успешно действующий агент должен сочетать в своем проекте и декларативные, и процедурные элементы.

Кроме передачи основанному на знаниях агенту с помощью операции `Tell` той информации, которую ему требуется знать, может также потребоваться предоставить ему механизмы, позволяющие обучаться самостоятельно. Эти механизмы, которые рассматриваются в главе 18, обеспечивают получение общих знаний о среде с помощью ряда актов восприятия. Такие знания могут быть включены в базу знаний агента и использоваться для принятия решений. Благодаря этому агент может стать полностью автономным.

Все эти возможности по представлению, формированию рассуждению и обучению опираются на результаты в области теории и практики логики как отрасли математики, накопленные в течение многих столетий. Но прежде чем перейти к описанию теории и практики, создадим простой мир, на примере которого сможем проиллюстрировать их применение.

## 7.2. МИР ВАМПУСА

❖ **Мир вампуса** — это пещера, состоящая из залов, соединенных проходами. Где-то по этой пещере бродит вампус — страшный зверь, который поедает всех, кто входит в зал, где он находится. Вампус может быть убит агентом, но у агента есть только одна стрела. В некоторых залах есть бездонные ямы, в которые попадают все, кто проходит через эти залы (все кроме вампуса, который слишком велик для того, чтобы в них провалиться). Единственное, что утешает живущих в этой среде, — это возможность найти кучу золота. Хотя мир вампуса является довольно скромным с точки зрения современных стандартов компьютерных игр, он представляет собой превосходную испытательную среду для интеллектуальных агентов. Первым, кто предложил использовать мир вампуса для такой цели, был Майкл Генезерет.

Пример мира вампуса показан на рис. 7.1. Точное определение среды этой задачи, соответствующее требованиям, которые приведены в главе 2, дано с помощью следующего описания PEAS.

- **Показатели производительности.** За то, что агент находит золото, он получает +1000 очков, за то, что агент попадает в яму или его съедает вампус, ему назначается -1000 очков, он отдает -1 очко за каждое выполненное действие и -10 очков — за использование стрелы.
- **Среда.** Залы расположены в виде решетки 4×4. Агент всегда начинает движение с квадрата, обозначенного как [1, 1], и смотрит вправо. Местонахождения золота и вампуса выбираются случайным образом с равномерным распределением из числа квадратов, отличных от начального квадрата. Кроме того, в

каждом квадрате, отличном от начального, с вероятностью 0,2 может находиться яма.

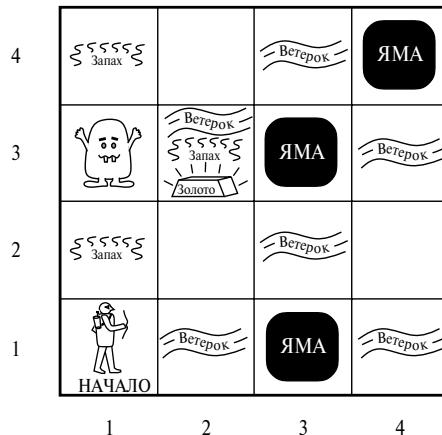


Рис. 7.1. Пример мира вампуса. Агент находится в нижнем левом углу

- **Исполнительные механизмы.** Агент может двигаться вперед, поворачиваться влево и вправо на 90°. Агент погибает жалкой смертью, если входит в квадрат, где имеется яма или живой вампус. (Входить в квадрат с мертвым вампусом безопасно, но при этом приходится чувствовать неприятный запах.) Попытка продвижения вперед остается безрезультатной, если перед агентом находится стена. Чтобы схватить объект, находящийся в том же квадрате, где находится агент, можно воспользоваться действием *Grab*. С помощью действия *Shoot* можно пустить стрелу по прямой линии в том направлении, куда смотрит агент. Стрела продолжает движение до тех пор, пока она либо не попадет в вампуса (и убьет его), либо не ударится в стену. У агента имеется только одна стрела, поэтому какой-либо эффект оказывает лишь первое действие *Shoot*.
- **Датчики.** У агента имеется пять датчиков, каждый из которых сообщает только один элемент информации:
  - в квадрате, где находится вампус, а также в квадратах, непосредственно (а не по диагонали) соседних по отношению к этому квадрату, агент чувствует неприятный запах (восприятие *Stench*);
  - в квадратах, непосредственно соседних с ямой, агент будет чувствовать ветерок (восприятие *Breeze*);
  - в квадрате, где находится золото, агент видит блеск (восприятие *Glitter*);
  - когда агент наталкивается на стену, он чувствует удар (восприятие *Bump*).
  - перед тем как пораженный стрелой вампус умирает, он испускает жалобный крик, который разносится по всей пещере (восприятие *Scream*).

Результаты актов восприятия передаются агенту в форме списка из пяти символов; например, если есть неприятный запах и ветерок, но нет блеска, удара или крика, агент воспринимает результаты акта восприятия [*Stench*, *Breeze*, *None*, *None*, *None*].

В упр. 7.1 предлагается определить среду вампуса в соответствии с различными измерениями, приведенными в главе 2. Основная сложность для данного агента состоит в том, что он с самого начала не знает конфигурацию своей среды; по-видимому, для преодоления этого незнания нельзя обойтись без логических рассуждений. В большинстве экземпляров мира вампуса для агента существует возможность безопасно получить золото. Но иногда агенту приходится выбирать: вернуться ли домой с пустыми руками или рисковать жизнью, чтобы найти золото. Около 21% вариантов среды являются совершенно неблагоприятными, поскольку золото находится в яме или окружено ямами.

Проследим за агентом для мира вампуса, действующим на основе знаний, который изучает среду, показанную на рис. 7.1. Первоначальная база знаний агента содержит правила существования в этой среде, которые были описаны выше; в частности, агент знает, что находится в квадрате [1, 1] и что квадрат [1, 1] является безопасным. Мы увидим, как расширяются знания агента по мере того, как поступают результаты новых актов восприятия и выполняются действия.

Первым восприятием является  $[None, None, None, None, None]$ , на основании которого агент может сделать вывод, что соседние по отношению к нему квадраты являются безопасными. На рис. 7.2, а показано состояние знаний агента в этот момент. Мы будем перечислять (некоторые) высказывания из базы знаний с использованием буквенных обозначений, таких как *B* (чувствуется ветерок) и *OK* (безопасно — ни ямы, ни вампуса), проставленных в соответствующих квадратах. С другой стороны, на рис. 7.1 изображен сам этот мир.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
<b>OK</b>			
1,1 [A] <b>OK</b>	2,1 <b>OK</b>	3,1	4,1

**A** - агент  
**B** - ветерок  
**G** - блеск, золото  
**OK** - безопасный квадрат  
**P** - яма  
**S** - запах  
**V** - посещенный квадрат  
**W** - вампус

a)
b)

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 <b>P?</b>	3,2	4,2
<b>OK</b>			
1,1 <b>V</b> <b>OK</b>	2,1 [A] <b>OK</b>	3,1 <b>P?</b>	4,1

Рис. 7.2. Первый шаг, выполненный агентом в мире вампуса: первоначальная ситуация, возникшая после восприятия  $[None, None, None, None, None]$  (а); ситуация после одного хода, в котором получено восприятие  $[None, Breeze, None, None, None]$  (б)

На основании того факта, что в квадрате [1, 1] не было ни неприятного запаха, ни ветерка, агент может сделать вывод, что квадраты [1, 2] и [2, 1] свободны от опасности. Для указания этого они отмечены обозначением *OK*. Осторожный агент переходит только в такой квадрат, о котором известно, что в нем есть отметка *OK*. Предположим, что агент решил двинуться вперед, в квадрат [2, 1], и была создана сцена, показанная на рис. 7.2, б.

Агент обнаруживает ветерок в квадрате [2, 1], поэтому в одном из соседних квадратов должна быть яма. По правилам игры яма не может находиться в квадрате [1, 1], поэтому она должна быть в квадрате [2, 2], или [3, 1], или в том и другом. Обозначение *P?* на рис. 7.2, б указывает на возможность наличия ямы в этих квадратах. В данный момент известен только один квадрат с отметкой *OK*, который еще не был посещен. Поэтому благоразумный агент поворачивается кругом и возвращается в квадрат [1, 1], а затем переходит в квадрат [1, 2].

Новым восприятием в квадрате [1, 2] является [*Stench*, *None*, *None*, *None*, *None*], что приводит к состоянию знаний, показанному на рис. 7.3, а. Наличие неприятного запаха в квадрате [1, 2] означает, что где-то рядом есть вампур. Но вампур не может находиться в квадрате [1, 1] по правилам игры и не может быть в квадрате [2, 2] (поскольку агент обнаружил бы неприятный запах, находясь в квадрате [2, 1]). Поэтому агент может сделать вывод, что вампур находится в квадрате [1, 3]. На это указывает обозначение *W!*. К тому же из отсутствия восприятия *Breeze* в квадрате [1, 2] следует, что в квадрате [2, 2] нет ямы. Тем не менее мы уже пришли к выводу, что яма должна быть в квадрате [2, 2] или [3, 1], а это означает, что она действительно находится в квадрате [3, 1]. Это — весьма сложный логический вывод, поскольку в нем объединяются знания, полученные в разное время в различных местах, и в нем решение о выполнении важного шага сделано на основании отсутствия определенного восприятия. Такой логический вывод превосходит способности большинства животных, но является типичным для рассуждений такого рода, которые выполняются логическими агентами.

1,4	2,4	3,4	4,4	
1,3 <b>W!</b>	2,3	3,3	4,3	
1,2 <b>A</b> <b>S</b> <b>OK</b>	2,2	3,2	4,2	
1,1 <b>V</b> <b>OK</b>	2,1 <b>B</b> <b>V</b> <b>OK</b>	3,1 <b>P!</b>	4,1	
				<b>A</b> - агент <b>B</b> - ветерок <b>G</b> - блеск, золото <b>OK</b> - безопасный квадрат <b>P</b> - яма <b>S</b> - запах <b>V</b> - посещенный квадрат <b>W</b> - вампур
1,4	2,4 <b>P?</b>	3,4	4,4	
1,3 <b>W!</b>	2,3 <b>A</b> <b>S</b> <b>G</b> <b>B</b>	3,3 <b>P?</b>	4,3	
1,2 <b>S</b> <b>V</b> <b>OK</b>	2,2	3,2	4,2	
1,1 <b>V</b> <b>OK</b>	2,1 <b>B</b> <b>V</b> <b>OK</b>	3,1 <b>P!</b>	4,1	

а)

б)

Рис. 7.3. Два последних этапа в ходе деятельности агента: после третьего хода, когда было получено восприятие [*Stench*, *None*, *None*, *None*, *None*] (а); после пятого хода, когда было получено восприятие [*Stench*, *Breeze*, *Glitter*, *None*, *None*] (б)

Теперь агент доказал сам себе, что в квадрате [2, 2] нет ни ямы, ни вампира, поэтому может обозначить этот квадрат меткой *OK*, чтобы в него перейти. Мы не показываем состояние знаний агента в квадрате [2, 2], а просто предполагаем, что агент повернулся и перешел в квадрат [2, 3], в результате чего было получено состояние, показанное на рис. 7.3, б. В квадрате [2, 3] агент обнаруживает блеск, поэтому должен схватить золото и тем самым закончить игру.

 В каждом случае, когда агент выводит заключение из доступной ему информации, гарантируется правильность этого заключения, если доступная информация является правильной. В этом состоит фундаментальное свойство логических рассуждений. В оставшейся части данной главы будет показано, как создавать логических агентов, которые могут представлять необходимую информацию и делать выводы, описанные в предыдущих абзацах.

### 7.3. ЛОГИКА

В этом разделе приведен краткий обзор всех фундаментальных понятий, касающихся логических представлений и рассуждений. Описание конкретных сведений о какой-либо определенной форме логики откладывается до следующего раздела. Вместо этого в данном разделе используются неформальные примеры из мира вампира и из знакомой всем области арифметики. Авторы приняли такой довольно необычный подход, поскольку идеи логики являются гораздо более универсальными и привлекательными, чем обычно предполагается.

В разделе 7.1 было указано, что базы знаний состоят из высказываний. Эти высказывания выражаются в соответствии с  **синтаксисом** языка представления, который определяет форму всех высказываний, рассматриваемых как построенные правильно. Понятие синтаксиса является достаточно очевидным в обычной арифметике: “ $x+y=4$ ” — правильно построенное высказывание, а “ $x2y+=$ ” — нет. Синтаксис логических языков (и арифметики в том числе) обычно формируется так, чтобы с его помощью было удобно писать статьи и книги. Существуют буквально десятки различных форм представления синтаксиса; в некоторых из них используются греческие буквы и сложные математические символы, а в других — довольно привлекательные на вид схемы со стрелками и кружками. Однако во всех случаях высказывания в базе знаний агента представляют собой реальные физические конфигурации (части) агента. Формирование рассуждений сводится к выработке этих конфигураций и манипулированию ими.

Логика должна также определять  **семантику** языка. Говоря неформально, семантика касается “смысла” высказываний. В логике это определение становится более точным. Семантика языка определяет  **истинность** каждого высказывания применительно к каждому из  **возможных миров**. Например, обычная семантика, принятая в арифметике, определяет, что высказывание “ $x+y=4$ ” истинно в мире, где  $x$  равно 2 и  $y$  равно 2, но ложно в мире, где  $x$  равно<sup>1</sup> 1 и  $y$  равно 1. В стандартной логике каждое высказывание должно быть либо истинным, либо ложным в каждом из возможных миров — в ней не может быть<sup>2</sup> каких-либо “промежуточных состояний”.

В дальнейшем, если потребуется уточнить какое-то определение, то вместо выражения один из “возможных миров” будет использоваться термин  **модель**. (Кроме того, для указания на то, что высказывание  $\alpha$  является истинным в модели  $m$ ,

<sup>1</sup> Несомненно, читатель заметил аналогию между понятием истинности высказываний и понятием удовлетворения ограничений, приведенным в главе 5. Это не случайно — языки ограничений действительно представляют собой языки логики, а решение ограничений — это одна из форм проведения логических рассуждений.

<sup>2</sup> В **нечеткой логике**, которая рассматривается в главе 14, допускается определение степени истинности.

будет использоваться фраза “ $t$  является моделью  $\alpha$ “.) Возможные миры могут рассматриваться как (потенциально) реальные варианты среды, в которых может находиться или не находиться агент, а модели представляют собой математические абстракции, каждая из которых устанавливает, является ли истинным или ложным каждое высказывание, относящееся к данной модели. Неформально, например, можно рассматривать  $x$  и  $y$  как количество мужчин и женщин, которые сидят за столом для игры в бридж, а высказывание  $x+y=4$  становится истинным, когда общее количество игроков равно четырем. В таком случае формально допустимые модели представляют собой все возможные присваивания целочисленных значений переменным  $x$  и  $y$ . Каждое такое присваивание устанавливает истинность любого высказывания в арифметике, переменными которого являются  $x$  и  $y$ .

Теперь, после рассмотрения понятия истинности, мы можем перейти к теме, касающейся логических рассуждений. Для этого необходимо определить понятие логического ~~и~~ следствия между высказываниями — идею о том, что одно высказывание может логически следовать из другого высказывания. В математических обозначениях применяется следующая запись:

$$\alpha \models \beta$$

которая означает, что высказывание  $\alpha$  влечет за собой высказывание  $\beta$ . Формальное определение логического следствия является таковым:  $\alpha \models \beta$  тогда и только тогда, когда в любой модели, в которой высказывание  $\alpha$  является истинным,  $\beta$  также истинно. Другой способ выразить эту мысль состоит в том, что если  $\alpha$  является истинным, высказывание  $\beta$  также является истинным. Неформально выражаясь, истинность высказывания  $\beta$  “содержится” в истинности высказывания  $\alpha$ . Понятие следствия применяется и в арифметике, например, из выражения  $x+y=4$  следует выражение  $4=x+y$ . Очевидно, что в любой модели, в которой  $x+y=4$  (например, в такой модели, в которой  $x$  равно 2 и  $y$  равно 2), справедливо также, что  $4=x+y$ . Вскоре будет показано, что любая база знаний может рассматриваться как некоторое утверждение, и в этой книге будет часто идти речь о том, что из некоторой базы знаний следует некоторое высказывание.

Анализ такого же типа можно применить и к примеру рассуждений о мире вампуса, приведенному в предыдущем разделе. Рассмотрим ситуацию, показанную на рис. 7.2, б: агент ничего не обнаружил в квадрате  $[1, 1]$ , а в квадрате  $[2, 1]$  почувствовал ветерок. Эти восприятия в сочетании со знаниями агента о правилах существования мира вампуса (описание PEAS на с. 286) составляют его базу знаний. Агенту необходимо узнать (кроме всего прочего), имеются ли ямы в соседних квадратах,  $[1, 2]$ ,  $[2, 2]$  и  $[3, 1]$ . В каждом из этих трех квадратов может находиться или не находиться яма, поэтому (в данном конкретном примере) существуют  $2^3=8$  возможных моделей. Они показаны<sup>3</sup> на рис. 7.4.

В моделях, которые противоречат тому, что знает агент, база знаний становится ложной; например, база знаний ложна в любой модели, в которой квадрат  $[1, 2]$  имеет яму, поскольку в квадрате  $[1, 1]$  не чувствуется ветерок. Существуют факти-

<sup>3</sup> Хотя на данном рисунке модели показаны как фрагменты мира вампуса, в действительности они представляют собой не что иное, как варианты присваивания значений `true` и `false` высказываниям “в квадрате  $[1, 2]$  имеется яма” и т.д. Для определения модели, в математическом смысле этого понятия, не нужно описывать в ней “страшных, покрытых шерстью” вампусов.

чески только три модели, в которых база знаний является истинной, и они показаны на рис. 7.4 как подмножество моделей. Теперь рассмотрим два возможных вывода:

$\alpha_1$  = "В квадрате [1, 2] нет ямы"

$\alpha_2$  = "В квадрате [2, 2] нет ямы"

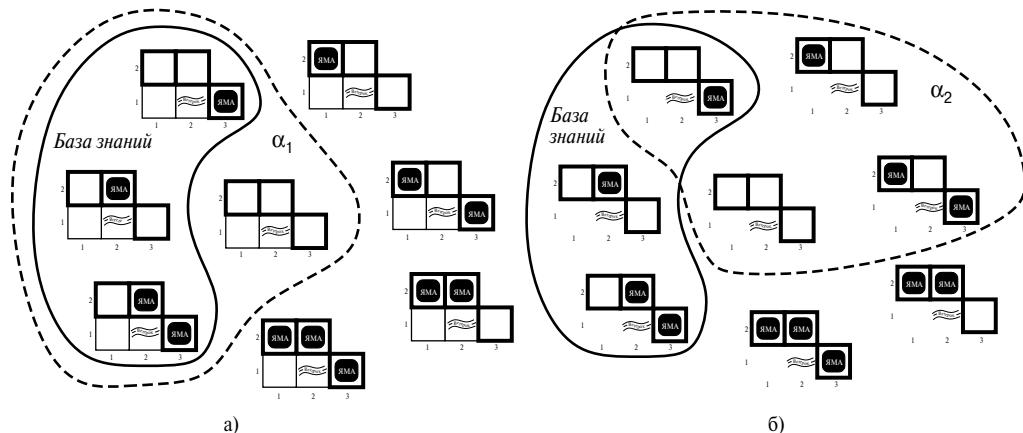


Рис. 7.4. Возможные модели, описывающие наличие ям в квадратах [1, 2], [2, 2] и [3, 1], которые основаны на тех наблюдениях, что в квадрате [1, 1] нет ничего, а в квадрате [2, 1] чувствуется ветерок: модели с базой знаний и высказыванием  $\alpha_1$  (в квадрате [1, 2] нет ямы) (а); модели с базой знаний и высказыванием  $\alpha_2$  (в квадрате [2, 2] нет ямы) (б)

Эти модели на рис. 7.4, а и б обозначены соответственно как  $\alpha_1$  и  $\alpha_2$ . Проанализировав все возможные ситуации, мы обнаружили следующее:

в каждой модели, в которой база знаний является истинной, высказывание  $\alpha_1$  также является истинным.

Поэтому  $KB \models \alpha_1$ : в квадрате [1, 2] нет ямы. Кроме того, можно убедиться в следующем:

в некоторых моделях, в которых база знаний является истинной, высказывание  $\alpha_2$  ложно.

Поэтому  $KB \not\models \alpha_2$ : агент не может прийти к заключению, что в квадрате [2, 2] нет ямы. (К тому же он не может сделать и вывод<sup>4</sup>, что в квадрате [2, 2] есть яма.)

В предыдущем примере не только проиллюстрирован процесс формирования логических следствий, но и показано, как можно применять определение логического следствия для формирования заключений, т.е. для проведения **логического вывода**. Алгоритм логического вывода, проиллюстрированный на рис. 7.4, называется **проверкой по моделям**, поскольку в нем осуществляется перебор всех возможных моделей для проверки того, что высказывание  $\alpha$  является истинным во всех моделях, в которых истинна база знаний.

Чтобы лучше понять, что такое логическое следствие и логический вывод, можно представить себе, что множество всех логических заключений, полученных из базы данных, — это стог сена, а высказывание  $\alpha$  — это иголка. Логическое следствие на-

<sup>4</sup> Агент может лишь вычислить вероятность того, что в квадрате [2, 2] есть яма; в главе 13 показано, как это сделать.

поминает утверждение о том, что в стоге сена есть иголка, а логический вывод можно сравнить с ее поиском. Это различие между понятиями логического следствия и логического вывода отражено и в формальных обозначениях: если некоторый алгоритм логического вывода  $i$  позволяет вывести логическим путем высказывание  $\alpha$  из базы знаний  $KB$ , то можно записать следующее:

$$KB \vdash_i \alpha$$

Эта формула читается так: “высказывание  $\alpha$  получено путем логического вывода из базы знаний  $KB$  с помощью алгоритма  $i$ ” или “алгоритм  $i$  позволяет вывести логическим путем высказывание  $\alpha$  из базы знаний  $KB$ ”.

Алгоритм логического вывода, позволяющий получить только такие высказывания, которые действительно следуют из базы знаний, называется **непротиворечивым**, или **сохраняющим истинность**. Непротиворечивость — это в высшей степени желательное свойство. Противоречивая процедура логического вывода в ходе своей работы по сути создает то, чего нет на самом деле: объявляет об обнаружении несуществующих иголок. Можно легко показать, что алгоритм проверки по моделям, если он применим<sup>5</sup>, является непротиворечивым.

Желательным является также свойство **полноты**: алгоритм логического вывода называется полным, если позволяет вывести любое высказывание, которое следует из базы знаний. Когда речь идет о настоящих стогах сена, имеющих конечный объем, то представляется вполне очевидным, что систематическое исследование всегда позволяет определить, есть ли иголка в данном стоге сена. Но со многими базами знаний связан бесконечно большой стог сена, состоящий из следствий, поэтому обеспечение полноты становится важной проблемой<sup>6</sup>. К счастью, существуют полные процедуры логического вывода для многих форм логики, которые являются достаточно выразительными для того, чтобы они могли справиться со многими базами знаний.

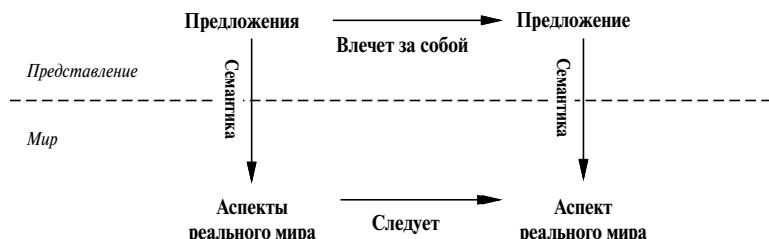
Выше фактически был описан процесс формирования рассуждений, выводы которого гарантированно являются истинными в любом мире, в котором истинны предпосылки; в частности, *если база знаний является истинной в реальном мире, то любое высказывание  $\alpha$ , полученное логическим путем из этой базы знаний с помощью непротиворечивой процедуры логического вывода, является также истинным в реальном мире*. Итак, несмотря на то, что процесс логического вывода оперирует с “синтаксисом” (а фактически с внутренними физическими конфигурациями, такими как биты в регистрах или картины электрических возбуждений в мозговых структурах), этот процесс соответствует связям реального мира, согласно которым некоторые аспекты реального мира имеют место<sup>7</sup> благодаря тому, что имеют место другие аспекты реального мира. Это соответствие между миром и его представлением продемонстрировано на рис. 7.5.

---

<sup>5</sup> Алгоритм проверки по моделям может применяться, если пространство моделей является конечным, например в мириях вампуса с постоянными размерами. В арифметике, с другой стороны, пространство моделей является бесконечным: даже если мы ограничимся целыми числами, то количество пар значений для  $x$  и  $y$  в выражении  $x+y=4$  является бесконечным.

<sup>6</sup> Сравните эту ситуацию со случаем бесконечных пространств поиска, описанным в главе 3, где поиск в глубину является неполным.

<sup>7</sup> Витгенштейн [1607] выразил эту мысль в своей знаменитой книге *Tractatus* (Трактат) таким образом: “Мир — это все, что имеет место”.



*Рис. 7.5. Высказывания представляют собой элементы физической конфигурации агента, а формирование рассуждений — это процесс создания новых элементов физической конфигурации из существующих. Процесс формирования логических рассуждений должен гарантировать, что новые элементы конфигурации будут представлять те аспекты мира, которые действительно следуют из аспектов, представленных существующими элементами конфигурации*

Последняя проблема, которая должна быть решена применительно к логическим агентам, — это проблема **обоснования**: установления связи между процессами логических рассуждений и реальной средой, в которой существует агент, если эта связь действительно имеется. В частности, эта проблема касается вопроса о том, **как узнать, что база знаний является истинной в реальном мире?** (В конце концов, ведь база знаний — просто “синтаксические конструкции” в голове агента.) Это — философская проблема, которой посвящено очень много книг (см. главу 26). Простой ответ состоит в том, что указанная связь создается с помощью датчиков агента. Например, наш агент для мира вампуса имеет датчик запаха. Обнаружив неприятный запах, программа агента создает соответствующее высказывание. Это означает, что если данное высказывание действительно находится в базе знаний, то является истинным и в реальном мире. Поэтому смысл и истинность высказываний, в которых выражаются результаты восприятий, определяются с помощью процессов формирования ощущений и составления высказываний, которые ими порождаются. А что можно сказать об остальных знаниях агента, который, например, считает, что вампуса можно обнаруживать из соседних квадратов, куда доносится его неприятный запах? Это — не прямое представление единственного акта восприятия, а общее правило, которое, возможно, было выведено из опыта восприятий, но не идентично утверждениям с описанием этого опыта. Выработка подобных общих правил в процессе формирования высказываний называется **обучением** и является темой части VI данной книги. Обучение чревато ошибками. Может оказаться, что вампусы распространяют неприятный запах во все дни, кроме 29 февраля високосного года, поскольку в эти дни вампусы принимают ванну. Поэтому в реальном мире база знаний может оказаться не истинной, но если агент оснащен хорошими процедурами обучения, у него есть повод для оптимизма.

## 7.4. ПРОПОЗИЦИОНАЛЬНАЯ ЛОГИКА: ОЧЕНЬ ПРОСТАЯ ЛОГИКА

В этом разделе будет представлена очень простая логика, называемая **пропозициональной логикой**<sup>8</sup>. Здесь рассматривается синтаксис пропозициональной логики.

<sup>8</sup> Пропозициональная логика называется также **булевой логикой** в честь логика Джорджа Буля (1815–1864).

гики и ее семантика — способ определения истинности высказываний. Затем мы рассмотрим понятие **следствия** (отношения между одним высказыванием и другим высказыванием, которое следует из него) и определим, как с помощью этого понятия можно получить простой алгоритм для логического вывода. Безусловно, все это происходит в мире вампуша.

## Синтаксис

**Синтаксис** пропозициональной логики определяет допустимые высказывания. **Атомарные высказывания** (неделимые синтаксические элементы) состоят из одного **пропозиционального символа**. Каждый такой символ обозначает высказывание, которое может быть либо истинным, либо ложным. Для обозначения подобных символов в данном разделе используются прописные буквы:  $P$ ,  $Q$ ,  $R$  и т.д. Эти обозначения являются произвольными, но часто выбираются таким образом, чтобы они имели для читателя какое-то мнемоническое значение. Например, символ  $W_{1,3}$  может использоваться для обозначения высказывания, согласно которому вампуш находится в квадрате [1, 3]. (Напомним, что символы, подобные  $W_{1,3}$ , являются атомарными; это означает, что  $W$ , 1 и 3 не следует рассматривать как осмысленные части этого символа.) Существуют два пропозициональных символа, имеющих постоянный смысл: *True* — тождественно истинное высказывание, а *False* — тождественно ложное высказывание.

**Сложные высказывания** формируются из более простых высказываний с помощью **логических связок**. Широко применяются пять описанных ниже логических связок.

- $\neg$  (нет). Такое высказывание, как  $\neg W_{1,3}$ , называется **отрицанием** высказывания  $W_{1,3}$ . **Литерал** представляет собой либо атомарное высказывание (**положительный литерал**), либо отрицаемое атомарное высказывание (**отрицательный литерал**).
- $\wedge$  (и). Высказывание, основной связкой которого является  $\wedge$ , такое как  $W_{1,3} \wedge P_{3,1}$ , называется **конъюнкцией**; его части называются **конъюнктами**. (Символ  $\wedge$  напоминает букву “A” в слове “And” — “И”.)
- $\vee$  (или). Высказывание, в котором используется связка  $\vee$ , такое как  $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$ , называется **дизъюнкцией дизъюнктов** ( $W_{1,3} \wedge P_{3,1}$  и  $W_{2,2}$ ). (Исторически обозначение  $\vee$  произошло из латинского слова “vel”, которое означает “или”. Большинство людей находят, что форму этой связки проще всего запомнить как перевернутый символ  $\wedge$ .)
- $\Rightarrow$  (влечет за собой). Высказывание, как  $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ , называется **импликацией** (или условным высказыванием). Его **предпосылкой**, или **антecedентом**, является  $(W_{1,3} \wedge P_{3,1})$ , а его **заключением**, или **консеквентом**, является  $\neg W_{2,2}$ . Импликации называют также **правилами**, или утверждениями **if-then** (если-то). В других книгах символ импликации иногда записывается как  $\supset$  или  $\rightarrow$ .
- $\Leftrightarrow$  (если и только если). Высказывание наподобие  $W_{1,3} \Leftrightarrow W_{2,2}$  называется **двухсторонней импликацией**.

Формальная грамматика пропозициональной логики показана в листинге 7.2; те, кто не знаком с системой обозначений BNF, должны обратиться за дополнительными сведениями на с. 1297.

### Листинг 7.2. Грамматика высказываний пропозициональной логики в форме BNF (Backus-Naur Form — форма Бэкуса–Наура)

---

```

Sentence → AtomicSentence | ComplexSentence
AtomicSentence → True | False | Symbol
Symbol → P | Q | R | ...
ComplexSentence → ¬Sentence
| ( Sentence ∧ Sentence )
| ( Sentence ∨ Sentence )
| ( Sentence ⇒ Sentence )
| ( Sentence ⇔ Sentence )

```

---

Обратите внимание на то, что эта грамматика предъявляет строгие требования к использованию круглых скобок: каждое высказывание, сформированное с помощью бинарных связок, должно быть заключено в круглые скобки. Это гарантирует полную непротиворечивость синтаксиса. Такое требование также означает, что следует писать, например,  $((A \wedge B) \Rightarrow C)$ , например, вместо  $A \wedge B \Rightarrow C$ . Но для удобства чтения мы будем часто опускать круглые скобки, полагаясь вместо них на использование порядка предшествования связок. Это аналогично правилам предшествования, используемым в арифметике, например, выражение  $ab+c$  читается как  $((ab)+c)$ , а не как  $a(b+c)$ , поскольку операция умножения имеет более высокий приоритет, чем сложение. Порядок предшествования в пропозициональной логике (от высшему к низшему) состоит в следующем:  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$  и  $\Leftrightarrow$ . Поэтому высказывание

$\neg P \vee Q \wedge R \Rightarrow S$

эквивалентно высказыванию

$((\neg P) \vee (Q \wedge R)) \Rightarrow S$

Определение порядка приоритета не позволяет устраниТЬ неоднозначность при чтении таких высказываний, как  $A \wedge B \wedge C$ , которое может быть прочитано как  $((A \wedge B) \wedge C)$  или  $(A \wedge (B \wedge C))$ . Но поскольку эти два прочтения, согласно семантике, описанной в следующем разделе, означают одно и то же, допускаются высказывания, подобные  $A \wedge B \wedge C$ . Разрешаются также высказывания наподобие  $A \vee B \vee C$  и  $A \Leftrightarrow B \Leftrightarrow C$ . А такие высказывания, как  $A \Rightarrow B \Rightarrow C$ , не допускаются, поскольку для них соответствующие два прочтения имеют разный смысл; мы настаиваем на том, что в этом случае должны использоваться круглые скобки. Наконец, в данной книге иногда вместо круглых скобок используются квадратные, если это позволяет немного упростить понимание данного высказывания.

## Семантика

Определив синтаксис пропозициональной логики, мы можем приступить к определению ее семантики. Семантика диктует правила выявления истинности высказывания по отношению к конкретной модели. В пропозициональной логике любая

модель просто фиксирует истинностное значение (*true* или *false*) для каждого пропозиционального символа. Например, если в высказываниях некоторой базы знаний используются пропозициональные символы  $P_{1,2}$ ,  $P_{2,2}$  и  $P_{3,1}$ , то одна из возможных моделей состоит в следующем:

$$m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}$$

При наличии трех пропозициональных символов существует  $2^3=8$  возможных моделей; именно столько моделей показано на рис. 7.4. Однако следует отметить, что с тех пор, как мы определили синтаксис, модели стали чисто математическими объектами, которые не обязательно должны быть связаны с миром вампуша. Например,  $P_{1,2}$  — это просто символ; он может означать, что “в квадрате [1, 2] есть яма” или “я буду в Париже сегодня и завтра”.

Семантика пропозициональной логики должна определять, как следует вычислять истинностное значение любого высказывания при наличии модели. Эта процедура выполняется рекурсивно. Все высказывания формируются из атомарных высказываний и пяти связок, поэтому необходимо указать, как следует вычислять истинность атомарных высказываний, а затем — как вычислять истинность высказываний, сформированных с помощью каждой из этих пяти связок. Задача вычисления истинности атомарных высказываний, как показано ниже, является простой.

- Высказывание *True* истинно в любой модели, а высказывание *False* ложно в любой модели.
- Истинностное значение любого другого пропозиционального символа должно быть указано непосредственно в модели. Например, в модели  $m_1$  приведенное выше высказывание  $P_{1,2}$  является ложным.

Для определения истинности сложных высказываний применяются правила, подобные приведенному ниже.

- Для любого высказывания  $s$  и любой модели  $m$  высказывание  $\neg s$  является истинным в модели  $m$  тогда и только тогда, когда  $s$  является ложным в модели  $m$ .

Эти правила позволяют свести задачу определения истинности сложных высказываний к задаче определения истинности более простых высказываний. Правила определения истинности для каждой связки могут быть подытожены в виде **истинностной таблицы**, которая определяет истинностное значение сложного высказывания для каждого возможного присваивания значений истинности его компонентам. Истинностные таблицы для рассматриваемых пяти логических связок приведены в табл. 7.1. С использованием этих таблиц истинностное значение любого высказывания  $s$  применительно к любой модели  $m$  может быть вычислено с помощью простого процесса рекурсивной оценки. Например, высказывание  $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$ , оцениваемое в модели  $m_1$ , приводит к получению  $\text{true} \wedge (\text{false} \vee \text{true}) = \text{true} \wedge \text{true} = \text{true}$ . В упр. 7.3 предложено написать алгоритм  $\text{PL-True?}(s, m)$ , который вычисляет истинностное значение любого высказывания  $s$  пропозициональной логики в модели  $m$ .

Выше было сказано, что любая база знаний состоит из множества высказываний. Теперь можно показать, что логическая база знаний представляет собой конъюнкцию этих высказываний. Это означает, что, начиная с пустой базы знаний  $KB$  и применяя операции  $\text{Tell}(KB, S_1), \dots, \text{Tell}(KB, S_n)$ , мы получим:  $KB = S_1 \wedge \dots \wedge S_n$ .

Таким образом, базы знаний и высказывания могут рассматриваться как взаимозаменяемые понятия.

**Таблица 7.1. Истинностные таблицы для пяти логических связок.** Чтобы воспользоваться этой таблицей, например для вычисления значения  $P \vee Q$ , когда  $P$  является истинным, а  $Q$  — ложным, вначале необходимо найти в левой части таблицы строку, в которой  $P$  имеет значение `true`, а  $Q$  — `false` (третья строка). Затем нужно искать запись в этой строке, соответствующую столбцу  $P \vee Q$ , чтобы определить результат — `true`. Еще один способ поиска истинностного значения состоит в том, что каждая строка может рассматриваться как модель, а записи под каждым столбцом для этой строки — как утверждение о том, является ли соответствующее высказывание истинным в данной модели

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<code>false</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>false</code>	<code>true</code>	<code>true</code>
<code>false</code>	<code>true</code>	<code>true</code>	<code>false</code>	<code>true</code>	<code>true</code>	<code>false</code>
<code>true</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>false</code>
<code>true</code>	<code>true</code>	<code>false</code>	<code>true</code>	<code>true</code>	<code>true</code>	<code>true</code>

Истинностные таблицы для связок  $\wedge$  (“и”),  $\vee$  (“или”) и  $\neg$  (“нет”) почти полностью соответствуют интуитивным представлениям о смысле таких же слов естественного языка. Основным источником возможной путаницы является то, что высказывание  $P \vee Q$  истинно, если истинными являются  $P$ , или  $Q$ , или оба эти высказывания. Есть также другая связка, называемая “исключительное ИЛИ” (сокращенно “XOR” — exclusive OR), которая принимает ложное значение, если оба дизъюнкта являются истинными<sup>9</sup>. В отношении того, какое обозначение следует применять для связки “исключительное ИЛИ”, нет общего согласия; двумя возможными вариантами являются  $\vee$  и  $\oplus$ .

Истинностная таблица для связки  $\Rightarrow$  на первый взгляд может показаться озадачивающей, поскольку не совсем соответствует интуитивному пониманию выражений “ $P$  влечет за собой  $Q$ ”, или “если  $P$ , то  $Q$ ”. Но прежде всего необходимо отметить, что пропозициональная логика не требует, чтобы между высказываниями  $P$  и  $Q$  устанавливались какие-либо причинно-следственные отношения или отношения, определяющие их релевантность применительно к друг другу. Высказывание “то, что 5 — нечетное число, влечет за собой то, что Токио — столица Японии”, — это истинное высказывание пропозициональной логики (при нормальной интерпретации), даже несмотря на то, что в естественном языке определенно кажется странным. Еще один источник путаницы состоит в том, что любая импликация является истинной, если ее антecedент ложен. Например, высказывание “то, что 5 — четное число, влечет за собой что, что Сэм умен”, является истинным, независимо от того, умен ли Сэм. Это может показаться странным, но приобретает смысл, если рассматривать высказывание в форме “ $P \Rightarrow Q$ ” как утверждение: “Если  $P$  истинно, то я утверждаю, что  $Q$  истинно. В противном случае я не высказываю никаких утверждений”. Единственный вариант, при котором это высказывание может принять значение `false`, состоит в том, чтобы высказывание  $P$  было истинным, а  $Q$  — ложным.

Истинностная таблица для двухсторонней импликации,  $P \Leftrightarrow Q$ , показывает, что это высказывание является истинным, если истинны и  $P \Rightarrow Q$ , и  $Q \Rightarrow P$ . В словес-

<sup>9</sup> В латинском языке для обозначения понятия “исключительного или” имеется отдельное слово, aut.

ной форме соответствующее высказывание часто записывают следующим образом: “ $P$  тогда и только тогда, когда  $Q$ ”, или сокращенно “ $P$  ттогда  $Q$ ” (тогда — не опечатка). Правила для мира вампуса лучше всего записывать с помощью связки  $\Leftrightarrow$ . Например, в некотором квадрате чувствуется ветерок, если в соседнем квадрате имеется яма, а ветерок в некотором квадрате может чувствоватьться, только если в одном из соседних квадратов имеется яма. Поэтому нам потребуются двухсторонние импликации, такие как

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

где  $B_{1,1}$  означает, что в квадрате  $[1, 1]$  чувствуется ветерок. Обратите внимание на то, что следующая односторонняя импликация:

$$B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$$

в мире вампуса является истинной, но неполной. Она не позволяет исключить из рассмотрения модели, в которых высказывание  $B_{1,1}$  ложно и  $P_{1,2}$  истинно и которые нарушают правила мира вампуса. Эту мысль можно иначе выразить таким образом, что импликация требует наличия ям, если чувствуется ветерок, а двухсторонняя импликация требует также отсутствия ям, если ветерок не чувствуется.

## Простая база знаний

Теперь, после определения семантики пропозициональной логики, мы можем сформировать базу знаний для мира вампуса. Для упрощения будем рассматривать только ямы; случай, в котором рассматривается также сам вампус, оставляем читателю в качестве упражнения. Мы предоставим агенту достаточный объем знаний, чтобы он мог сам формировать те логические выводы, которые были описаны неформально в разделе 7.3.

Вначале необходимо определить словарь пропозициональных символов. Для каждого  $i, j$ :

- допустим, что высказывание  $P_{i,j}$  является истинным, если в квадрате  $[i, j]$  имеется яма;
- допустим, что  $B_{i,j}$  является истинным, если в квадрате  $[i, j]$  чувствуется ветерок.

База знаний включает перечисленные ниже высказывания, каждому из которых для удобства присвоено отдельное обозначение.

- В квадрате  $[1, 1]$  отсутствует яма:

$$R_1: \neg P_{1,1}$$

- В квадрате чувствуется ветерок тогда и только тогда, когда в соседнем квадрате имеется яма. Такое высказывание должно быть сформулировано для каждого квадрата; на данный момент включены в рассмотрение только непосредственно интересующие нас квадраты:

$$R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

- Приведенные выше высказывания являются истинными во всех экземплярах мира вампуса. Теперь включим данные о восприятии ветерка для первых двух

квадратов, которые были посещены агентом в том конкретном мире, где он находится; это приведет нас к ситуации, показанной на рис. 7.2, б.

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$

Таким образом, база знаний состоит из высказываний  $R_1-R_5$ . Ее можно также рассматривать как единственное высказывание (как конъюнкцию  $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$ ), поскольку она подтверждает, что все отдельно взятые высказывания в ней являются истинными.

### Логический вывод

Напомним, что цель логического вывода состоит в том, чтобы определить, является ли истинным выражение  $KB \models \alpha$  для некоторого высказывания  $\alpha$ . Например, следует ли из базы знаний высказывание  $P_{2,2}$ ? Рассматриваемый в данном разделе первый алгоритм логического вывода представляет собой непосредственную реализацию на практике определения логического следствия: перебрать (перечислить) все модели и проверить, является ли высказывание  $\alpha$  истинным в каждой модели, в которой база знаний  $KB$  является истинной. Для пропозициональной логики модели представляют собой варианты присваивания значений *true* или *false* каждому пропозициальному символу. Возвратившись к примеру с миром вампуса, определим, что соответствующими пропозициональными символами являются  $B_{1,1}$ ,  $B_{2,1}$ ,  $P_{1,1}$ ,  $P_{1,2}$ ,  $P_{2,1}$ ,  $P_{2,2}$  и  $P_{3,1}$ . При наличии семи символов могут существовать  $2^7=128$  возможных моделей; в трех из них база знаний  $KB$  является истинной (табл. 7.2). В этих трех моделях является также истинным высказывание  $\neg P_{1,2}$ , поэтому в квадрате [1, 2] нет ямы. С другой стороны, высказывание  $P_{2,2}$  истинно в двух из трех моделей и ложно в одной из них, поэтому мы еще не можем определить, имеется ли яма в квадрате [2, 2].

**Таблица 7.2. Истинностная таблица, которая сформирована для базы знаний, определенной в тексте главы. База знаний  $KB$  является истинной, если истинны высказывания  $R_1-R_5$ , а это происходит только в 3 из 128 строк. Во всех 3 строках высказывание  $P_{1,2}$  ложно, поэтому в квадрате [1, 2] нет ямы. С другой стороны, яма в квадрате [2, 2] может быть (или не быть)**

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$KB$
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>	
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>false</i>	
...	...	...	...	...	...	...	...	...	...	...	...	...
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	
...	...	...	...	...	...	...	...	...	...	...	...	...
<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>false</i>

В табл. 7.2 воспроизведен в более точной форме процесс формирования рассуждений, который проиллюстрирован на рис. 7.4. Общий алгоритм определения логи-

ческого следствия в пропозициональной логике приведен в листинге 7.3. Как и в алгоритме Backtracking-Search, приведенном на с. 131, функция TT-Entails? осуществляет рекурсивный перебор конечного пространства вариантов присваивания значений переменным. Этот алгоритм является **непротиворечивым**, поскольку он непосредственно реализует определение логического следствия, и **полным**, поскольку может применяться для любой базы знаний *KB* и любого высказывания  $\alpha$  и всегда заканчивает свою работу, при условии, что количество моделей, подлежащих проверке, является конечным.

**Листинг 7.3.** Алгоритм перебора истинностной таблицы для получения пропозициональных логических следствий. *TT* обозначает истинностную таблицу, функция *PL-True?* возвращает истинное значение, если некоторое высказывание является истинным в рамках некоторой модели. Переменная *model* представляет частично заданную модель — присваивание только некоторым переменным. Вызов функции *Extend(P, true, model)* возвращает новую частично заданную модель, в которой высказывание *P* имеет значение *true*

---

```

function TT-Entails?(KB,  $\alpha$ ) returns значение true или false
  inputs: KB, база знаний — высказывание в пропозициональной логике
           $\alpha$ , запрос — высказывание в пропозициональной логике

  symbols  $\leftarrow$  список пропозициональных символов в KB и  $\alpha$ 
  return TT-Check-All(KB,  $\alpha$ , symbols, [])

function TT-Check-All(KB,  $\alpha$ , symbols, model) returns значение
  true или false
  if Empty?(symbols) then
    if PL-True?(KB, model) then return PL-True?( $\alpha$ , model)
    else return true
  else do
    P  $\leftarrow$  First(symbols); rest  $\leftarrow$  Rest(symbols)
    return TT-Check-All(KB,  $\alpha$ , rest, Extend(P, true, model) and
              TT-Check-All(KB,  $\alpha$ , rest, Extend(P, false, model))

```

---

Безусловно, выражение “является конечным” не всегда означает то же, что и выражение “является небольшим”. Если *KB* и  $\alpha$  содержат в целом  $n$  символов, то количество моделей равно  $2^n$ . Таким образом, времененная сложность этого алгоритма составляет  $O(2^n)$ . (Пространственная сложность составляет только  $O(n)$ , поскольку перебор вариантов присваивания происходит по принципу поиска в глубину.) Ниже будут показаны алгоритмы, которые являются гораздо более эффективными на практике. К сожалению, *каждый известный алгоритм логического вывода для пропозициональной логики в худшем случае имеет сложность, экспоненциально зависящую от размера ввода*. Мы не можем рассчитывать на то, что наши алгоритмы будут действовать лучше, поскольку задача получения пропозиционального логического следствия является ко-NP-полной (см. приложение А).

### Эквивалентность, допустимость и выполнимость

Прежде чем перейти к изложению подробных сведений об алгоритмах логического вывода, необходимо рассмотреть некоторые дополнительные понятия, ка-

сающиеся логического следствия. Как и само понятие следствия, эти понятия относятся ко всем формам логики, но их лучше всего показать на примере какого-то конкретного варианта логики, например пропозициональной логики.

Первым понятием является **логическая эквивалентность**: два высказывания,  $\alpha$  и  $\beta$ , являются логически эквивалентными, если они истинны в одном и том же множестве моделей. Это утверждение записывается как  $\alpha \Leftrightarrow \beta$ . Например, можно легко показать (с помощью истинностных таблиц), что высказывания  $P \wedge Q$  и  $Q \wedge P$  логически эквивалентны; другие эквивалентности приведены в листинге 7.4. Они играют в логике практически такую же роль, какую арифметические равенства играют в обычной математике. Альтернативное определение эквивалентности является следующим: для любых двух высказываний  $\alpha$  и  $\beta$

$$\alpha \equiv \beta \text{ тогда и только тогда, когда } \alpha \vDash \beta \text{ и } \beta \vDash \alpha$$

(Еще раз напоминаем, что  $\vDash$  означает логическое следствие.)

**Листинг 7.4. Стандартные логические эквивалентности. Символы  $\alpha$ ,  $\beta$  и  $\gamma$  обозначают произвольные высказывания пропозициональной логики**

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	коммутативность связки $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	коммутативность связки $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	ассоциативность связки $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	ассоциативность связки $\vee$
$\neg(\neg \alpha) \equiv \alpha$	удаление двойного отрицания
$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$	контрапозиция
$(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$	удаление импликации
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	удаление двухсторонней импликации
$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$	правило де Моргана
$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$	правило де Моргана
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	дистрибутивность связки $\wedge$ по $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	дистрибутивность связки $\vee$ по $\wedge$

Вторым понятием, которое нам потребуется, является **допустимость**. Высказывание допустимо, если оно истинно во всех моделях. Например, высказывание  $P \vee \neg P$  является допустимым. Допустимые высказывания известны также под названием **тавтологий** — они обязательно истинны и поэтому избыточны. Поскольку высказывание *True* является истинным во всех моделях, то любое допустимое высказывание логически эквивалентно *True*.

Для чего могут применяться допустимые высказывания? Из определения логического следствия можно вывести **теорему дедукции**, которая была известна еще в древней Греции:

 Для любых высказываний  $\alpha$  и  $\beta$ ,  $\alpha \vDash \beta$ , если и только если высказывание  $(\alpha \Rightarrow \beta)$  является допустимым.

(В упр. 7.4 предлагается доказать эту теорему.) Алгоритм логического вывода, приведенный в листинге 7.3, может рассматриваться как проверка допустимости высказывания ( $KB \Rightarrow \alpha$ ). И наоборот, каждое допустимое высказывание в форме импликации описывает приемлемый вариант логического вывода.

Последним понятием, которое нам потребуется, является **выполнимость**. Некоторое высказывание выполнимо тогда и только тогда, когда оно истинно в неко-

торой модели. Например, приведенная выше база знаний,  $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$ , выполнима, поскольку существует даже не одна, а три модели, в которых она истинна (см. табл. 7.2). Если некоторое высказывание  $\alpha$  является истинным в модели  $m$ , то для обозначения этого применяется выражение, что “ $m$   $\models$  выполняет  $\alpha$ ”, или что “ $m$  является моделью  $\alpha$ ”. Выполнимость можно проверить, перебирая все возможные модели до тех пор, пока не будет найдена хотя бы одна из них, которая выполняет данное высказывание. Первой задачей в пропозициональной логике, для которой было доказано, что она является NP-полной, была задача определения выполнимости высказываний.

Многие задачи в компьютерных науках в действительно представляют собой задачи определения выполнимости. Например, во всех задачах удовлетворения ограничений, приведенных в главе 5, по сути требовалось найти ответ на вопрос о том, выполнимы ли данные ограничения при некотором присваивании. Кроме того, с помощью соответствующих преобразований по методу проверки выполнимости можно решать и задачи поиска. Безусловно, что понятия допустимости и выполнимости связаны друг с другом: высказывание  $\alpha$  является допустимым тогда и только тогда, когда высказывание  $\neg\alpha$  невыполнимо, и наоборот, высказывание  $\alpha$  является выполнимым тогда и только тогда, когда высказывание  $\neg\alpha$  недопустимо. На этом основании может быть также получен следующий полезный результат:

 *Высказывание  $\alpha \models \beta$  истинно, если и только если высказывание  $(\alpha \wedge \neg\beta)$  невыполнимо.*

Доказательство истинности высказывания  $\beta$  на основании истинности высказывания  $\alpha$  путем проверки невыполнимости выражения  $(\alpha \wedge \neg\beta)$  точно соответствует стандартному математическому методу доказательства путем  $\models$  **приведения к абсурду** (буквально, “доведение до абсурда” — *reductio ad absurdum*). Его также называют доказательством путем  $\models$  **опровержения**, или доказательством с помощью **выявления противоречия**. В этом методе доказательства принимается предположение, что высказывание  $\beta$  ложно, и демонстрируется, что такое предположение приводит к противоречию с известными аксиомами  $\alpha$ . Подобное противоречие точно соответствует тому, что подразумевается под утверждением, что выражение  $(\alpha \wedge \neg\beta)$  невыполнимо.

## 7.5. ШАБЛОНЫ ФОРМИРОВАНИЯ РАССУЖДЕНИЙ В ПРОПОЗИЦИОНАЛЬНОЙ ЛОГИКЕ

В данном разделе рассматриваются стандартные шаблоны логического вывода, которые могут применяться для формирования цепочек заключений, ведущих к желаемой цели. Эти шаблоны логического вывода называются  $\models$  **правилами логического вывода**. Наиболее широко известное правило называется  $\models$  **правилом отщепления** (*Modus Ponens* — модус поненс) и записывается следующим образом:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

Эта запись означает, что если даны любые высказывания в форме  $\alpha \Rightarrow \beta$  и  $\alpha$ , то из них можно вывести высказывание  $\beta$ . Например, если дано  $(WumpusAhead \wedge$

$WumpusAlive) \Rightarrow Shoot$  и  $(WumpusAhead \wedge WumpusAlive)$ , то можно вывести высказывание  $Shoot$ .

Еще одним полезным правилом логического вывода является **правило удаления связки “И”**, в котором утверждается, что из конъюнкции можно вывести любой из ее конъюнктоов:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Например, из высказывания  $(WumpusAhead \wedge WumpusAlive)$  можно вывести высказывание  $WumpusAlive$ .

Рассматривая возможные истинностные значения  $\alpha$  и  $\beta$ , можно легко показать, что правило отделения и правило удаления связки “И” являются непротиворечивыми не только применительно к данным примерам, но и ко всем возможным высказываниям. Это означает, что данные правила могут использоваться во всех конкретных случаях, в которых они могут потребоваться, вырабатывая непротиворечивые логические выводы без необходимости перебирать все модели.

В качестве правил логического вывода можно также применять все логические эквивалентности, приведенные в листинге 7.4. Например, из эквивалентности двухсторонней импликации и двух импликаций можно получить следующие два правила логического вывода (первое из них называется *правилом удаления двухсторонней импликации*):

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \text{ и } \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Но не все правила логического вывода действуют в обоих направлениях, как это. Например, нельзя применить правило отделения в противоположном направлении, чтобы получить высказывания  $\alpha \Rightarrow \beta$  и  $\alpha$  из высказывания  $\beta$ .

Рассмотрим, как можно использовать эти правила логического вывода и эквивалентности в мире вампуша. Начнем с базы знаний, содержащей высказывания  $R_1 - R_5$ , и покажем, как доказать высказывание  $\neg P_{1,2}$ , т.е. утверждение, что в квадрате  $[1, 2]$  нет ямы. Вначале применим правило удаления двухсторонней импликации к высказыванию  $R_2$ , чтобы получить следующее:

$$R_6: (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

Затем применим к высказыванию  $R_6$  правило удаления связки “И” и получим:

$$R_7: ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

Из логической эквивалентности отрицаний следует:

$$R_8: (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$$

Теперь можно применить правило отделения к высказыванию  $R_8$  и к высказыванию  $R_4$  с данными восприятия (т.е.  $\neg B_{1,1}$ ), чтобы получить следующее:

$$R_9: \neg(P_{1,2} \vee P_{2,1})$$

Наконец, применим правило де Моргана, позволяющее получить такое заключение:

$$R_{10}: \neg P_{1,2} \wedge \neg P_{2,1}$$

Это означает, что ни в квадрате  $[1, 2]$ , ни в квадрате  $[2, 1]$  нет ямы.

Приведенный выше процесс логического вывода (последовательность применения правил логического вывода) называется **доказательством**. Процесс формирования доказательств полностью аналогичен процессу обнаружения решений в задачах поиска. В действительности, если можно было бы составить функцию определения преемника для выработки всех возможных вариантов применения правил логического вывода, это позволило бы использовать для формирования доказательств все алгоритмы поиска, приведенные в главах 3 и 4. Таким образом, вместо трудоемкого перебора моделей может применяться более эффективный способ логического вывода — формирование доказательств. Поиск может проходить в прямом направлении, от первоначальной базы знаний, и осуществляться путем использования правил логического вывода для получения целевого высказывания, или же этот поиск может проходить в обратном направлении, от целевого высказывания, в попытке найти цепочку правил логического вывода, ведущую от первоначальной базы знаний. Ниже в этом разделе рассматриваются два семейства алгоритмов, в которых используются оба эти подхода.

Из того факта, что задача логического вывода в пропозициональной логике является NP-полной, следует, что в наихудшем случае поиск доказательств может оказаться не более эффективным по сравнению с перебором моделей. Однако во многих практических случаях *поиск доказательства может быть чрезвычайно эффективным просто потому, что в нем могут игнорироваться не относящиеся к делу (нерелевантные) высказывания, независимо от того, насколько велико их количество*. Например, в приведенном выше доказательстве, ведущем к высказыванию  $\neg P_{1,2} \wedge \neg P_{2,1}$ , не упоминались высказывания  $B_{2,1}$ ,  $P_{1,1}$ ,  $P_{2,2}$  или  $P_{3,1}$ . Их можно было не рассматривать, поскольку целевое высказывание,  $P_{1,2}$ , присутствует только в высказывании  $R_4$ ; остальные высказывания из состава  $R_4$  присутствуют только в  $R_4$  и  $R_2$ , поэтому  $R_1$ ,  $R_3$  и  $R_5$  не имеют никакого отношения к доказательству. Те же рассуждения останутся справедливыми, если в базу знаний будет введено на миллион больше высказываний; с другой стороны, в этом случае простой алгоритм перебора строк в истинностной таблице был бы буквально подавлен из-за экспоненциального взрыва, вызванного стремительным увеличением количества моделей.

Это свойство логических систем фактически вытекает из гораздо более фундаментального свойства, называемого **монотонностью**. Согласно этому свойству, множество высказываний, которые могут быть получены путем логического вывода, возрастает лишь по мере добавления к базе знаний новой информации<sup>10</sup>. Для любых высказываний  $\alpha$  и  $\beta$  справедливо следующее:

$$\text{если } KB \models \alpha, \text{ то } KB \wedge \beta \models \alpha$$

Например, предположим, что база знаний содержит дополнительное утверждение  $\beta$ , согласно которому в этом экземпляре мира вампуса имеется точно восемь ям. Эти знания могут помочь агенту прийти к дополнительным заключениям, но не способны поставить под сомнение какое-либо уже сделанное заключение  $\alpha$ , в частности вывод о том, что в квадрате [1, 2] нет ямы. Монотонность означает, что правила логического вывода могут применяться каждый раз, когда в базе знаний обна-

<sup>10</sup> **Немонотонные** логики, в которых нарушается свойство монотонности, отражают типичную черту человеческого мышления — способность менять свое мнение. Эти варианты логики рассматриваются в разделе 10.7.

ружаются подходящие предпосылки, — полученное заключение будет следствием из данного правила, независимо от того, что еще находится в базе знаний.

## Резолюция

Выше было показано, что все рассматривавшиеся до сих пор правила логического вывода являются непротиворечивыми, но вопрос об их полноте применительно к алгоритмам логического вывода, в которых они используются, еще не обсуждался. Алгоритмы поиска, такие как поиск с итеративным углублением (с. 133), являются полными в том смысле, что позволяют найти любую достижимую цель, но если доступные правила логического вывода недекватны, то цель становится недостижимой; это означает, что не существует доказательства, в котором могли бы применяться только эти правила логического вывода. Например, если мы откажемся от использования правила удаления двухсторонней импликации, то не сможем довести до конца доказательство, изложенное в предыдущем разделе. А в настоящем разделе представлено единственное правило логического вывода, **правило резолюции**, позволяющее получить алгоритм логического вывода, который становится полным в сочетании с любым полным алгоритмом поиска.

Начнем с использования простой версии правила резолюции в мире вампуса. Рассмотрим шаги, ведущие вверх на рис. 7.3, а: агент возвращается из квадрата  $[2, 1]$  в квадрат  $[1, 1]$ , а затем переходит в квадрат  $[1, 2]$ , где он чувствует неприятный запах, но не ощущает ветерка. Введем следующие дополнительные факты в базу знаний:

$$\begin{aligned} R_{11}: \quad & \neg B_{1,2} \\ R_{12}: \quad & B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3}) \end{aligned}$$

С помощью того же процесса, который привел к получению высказывания  $R_{10}$ , приведенного выше, мы теперь можем сделать заключение об отсутствии ям в квадратах  $[2, 2]$  и  $[1, 3]$  (напомним, что в отношении квадрата  $[1, 1]$  уже известно, что в нем нет ям):

$$\begin{aligned} R_{13}: \quad & \neg P_{2,2} \\ R_{14}: \quad & \neg P_{1,3} \end{aligned}$$

Кроме того, можно применить к высказыванию  $R_3$  правило удаления двухсторонней импликации, после чего применить к полученному результату в сочетании с высказыванием  $R_5$  правило отделения, чтобы выяснить тот факт, что по меньшей мере в одном из квадратов  $[1, 1]$ ,  $[2, 2]$  или  $[3, 1]$  есть яма:

$$R_{15}: \quad P_{1,1} \vee P_{2,2} \vee P_{3,1}$$

Именно здесь возникают условия, позволяющие впервые использовать правило резолюции (правило устранения противоречия): литерал  $\neg P_{2,2}$  в высказывании  $R_{13}$ , противоположный литералу  $P_{2,2}$  в высказывании  $R_{15}$ , устраняется, что приводит к получению следующего высказывания:

$$R_{16}: \quad P_{1,1} \vee P_{3,1}$$

Соответствующий ход рассуждения можно выразить словами таким образом: если по меньшей мере в одном из квадратов  $[1, 1]$ ,  $[2, 2]$  или  $[3, 1]$  имеется яма, но ее нет в квадрате  $[2, 2]$ , то яма должна быть, по крайней мере, в квадрате  $[1, 1]$

или [3, 1]. Аналогичным образом, устраняется литерал  $\neg P_{1,1}$  в высказывании  $R_1$ , противоположный литералу  $P_{1,1}$  в высказывании  $R_{16}$ , что приводит к получению следующего высказывания:

$$R_{17}: \quad P_{3,1}$$

Соответствующий ход рассуждений можно выразить словами таким образом: если по меньшей мере в одном из квадратов [1, 1] или [3, 1] есть яма, но ее нет в квадрате [1, 1], то она находится в квадрате [3, 1]. Эти два последних этапа логического вывода представляют собой примеры применения **правила логического вывода**, называемого правилом **единичной резолюции**:

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k}$$

где каждое из выражений  $\ell$  представляет собой литерал, а выражения  $\ell_i$  и  $m$  являются **взаимно обратными литералами** (т.е. такими литералами, что один из них является отрицанием другого). Таким образом, в правиле единичной резолюции берется **выражение** (дизъюнкция литералов) и еще один литерал, после чего формируется новое выражение. Обратите внимание на то, что этот единственный литерал может рассматриваться как дизъюнкция из одного литерала, называемая также **единичным выражением**.

Правило единичной резолюции может быть также обобщено до правила полной **резолюции**:

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m_1 \vee \dots \vee m_n}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

где  $\ell_i$  и  $m_j$  — взаимно обратные литералы. Если бы мы имели дело только с выражениями, имеющими длину два, то могли бы записать данное правило следующим образом:

$$\frac{\ell_1 \vee \ell_2, \quad \neg \ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Это означает, что в правиле резолюции берутся два выражения и вырабатывается новое выражение, содержащее все литералы двух первоначальных выражений, за исключением двух взаимно обратных литералов. Например, может иметь место такой логический вывод:

$$\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Процесс применения правила резолюции связан с необходимостью выполнения еще одного формального требования: результирующее высказывание должно содержать только по одной копии каждого литерала<sup>11</sup>. Операция удаления дополнитель-

<sup>11</sup> Если некоторое выражение рассматривается как множество литералов, то такое правило сокращения применяется автоматически. Использование для логических выражений системы обозначений в виде множеств позволяет сделать правило резолюции более понятным, но за счет введения дополнительных обозначений.

ных копий литералов называется **факторизацией**. Например, после удаления противоположных друг другу литералов в выражениях  $(A \vee B)$  и  $(A \vee \neg B)$  будет получено выражение  $(A \vee A)$ , которое следует сократить до  $A$ .

Непротиворечивость правила резолюции можно легко доказать, рассматривая литерал  $\ell_i$ . Если литерал  $\ell_i$  является истинным, то литерал  $m_j$  является ложным, и поэтому выражение  $m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n$  должно быть истинным, поскольку дано, что истинно выражение  $m_1 \vee \dots \vee m_n$ . Если литерал  $\ell_i$  является ложным, то должно быть истинным выражение  $\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k$ , поскольку дано, что истинно выражение  $\ell_1 \vee \dots \vee \ell_k$ . Итак, литерал  $\ell_i$  является либо истинным, либо ложным, поэтому справедливо одно или второе из этих заключений, а именно это утверждается в правиле резолюции.

Еще более привлекательным свойством правила резолюции является то, что оно образует основу для семейства полных процедур логического вывода. **Любой полный алгоритм поиска, в котором применяется только правило резолюции, позволяет вывести любое заключение, которое следует из любой базы знаний в пропозициональной логике.** Тем не менее необходимо сделать одно предупреждение: правило резолюции является полным только в узком смысле этого понятия. Если известно, что высказывание  $A$  истинно, правило резолюции нельзя использовать для автоматического формирования следствия  $A \vee B$ . Однако в этом случае правилом резолюции можно воспользоваться для поиска ответа на вопрос, является ли истинным высказывание  $A \vee B$ . Это свойство правила резолюции называется **полнотой опровержения** (refutation completeness) и означает, что правило резолюции может всегда использоваться либо для подтверждения, либо для опровержения какого-то высказывания, но его нельзя применять для перебора всех истинных высказываний. В следующих двух подразделах описано, как может использоваться правило резолюции для осуществления логического вывода.

### Конъюнктивная нормальная форма

Как было описано выше, правило резолюции применяется только к дизъюнциям литералов, поэтому на первый взгляд оно распространяется только на базы знаний и запросы, состоящие из таких дизъюнкций. На каком же основании мы утверждаем, что это правило может служить основой процедуры полного логического вывода для всей пропозициональной логики? Ответ на этот вопрос состоит в том, что **каждое высказывание пропозициональной логики логически эквивалентно конъюнкции дизъюнкций литералов**. Любое высказывание, представленное как конъюнкция дизъюнкций литералов, называется высказыванием, находящимся в **конъюнктивной нормальной форме**, или **CNF (Conjunctive Normal Form)**. Кроме того, ниже будет показано, что целесообразно определить также ограниченное семейство высказываний в конъюнктивной нормальной форме, называемое высказываниями в форме **k-CNF**. Высказывание в форме k-CNF имеет точно  $k$  литералов в расчете на каждое выражение:

$$(\ell_{1,1} \vee \dots \vee \ell_{1,k}) \wedge \dots \wedge (\ell_{n,1} \vee \dots \vee \ell_{n,k})$$

Как оказалось, любое высказывание может быть преобразовано в высказывание в форме 3-CNF, которое имеет эквивалентное множество моделей.

Вместо доказательства этих утверждений (см. упр. 7.10) опишем простую процедуру преобразования. Проиллюстрируем эту процедуру, преобразовав высказывание  $R_2$ , или  $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ , в форму CNF. Ниже описаны соответствующие этапы.

1. УстраниТЬ связку  $\Leftrightarrow$ , заменив высказывание  $\alpha \Leftrightarrow \beta$  высказыванием  $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ :  

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$
2. УстраниТЬ связку  $\Rightarrow$ , заменив высказывание  $\alpha \Rightarrow \beta$  высказыванием  $\neg\alpha \vee \beta$ :  

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$
3. Конъюнктивная нормальная форма требует, чтобы связка  $\neg$  появлялась только перед литералами, поэтому, как принято называть эту операцию, “введем связку  $\neg$  внутрь выражения”, повторяя операцию применения следующих эквивалентностей из листинга 7.4:

$$\begin{array}{ll} \neg(\neg \alpha) \equiv \alpha & \text{(удаление двойного отрицания)} \\ \neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) & \text{(правило де Моргана)} \\ \neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) & \text{(правило де Моргана)} \end{array}$$

В данном примере требуется применение только одного, последнего правила:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. В результате получено высказывание, содержащее вложенные связки  $\wedge$  и  $\vee$ , которые применяются к литералам. Используем закон дистрибутивности, приведенный в листинге 7.4, распределяя связки  $\vee$  по связкам  $\wedge$  везде, где это возможно.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Теперь первоначальное высказывание представлено в форме CNF, как конъюнкция трех выражений. Это высказывание стало более сложным для чтения, но зато его можно использовать в качестве входных данных для процедуры резолюции.

### Алгоритм резолюции

Процедуры логического вывода, основанные на правиле резолюции, действуют с использованием принципа доказательства путем установления противоречия, который описывался в конце раздела 7.4. Таким образом, чтобы показать, что  $KB \models \alpha$ , мы покажем, что высказывание  $(KB \wedge \neg\alpha)$  является невыполнимым. Это можно сделать, доказав, что имеет место противоречие.

Алгоритм резолюции приведен в листинге 7.5. Вначале высказывание  $(KB \wedge \neg\alpha)$  преобразуется в форму CNF. Затем к результирующим выражениям применяется правило резолюции. В каждой паре выражений, содержащих взаимно противоположные литералы, происходит удаление этих противоположных друг другу литералов для получения нового выражения, которое добавляется к множеству существующих выражений, если в этом множестве еще нет такого выражения. Указанный процесс продолжается до тех пор, пока не происходит одно из следующих двух событий:

- перестают вырабатываться какие-либо новые выражения, которые могли быть добавлены к существующим, и в этом случае из базы знаний  $KB$  не следует высказывание  $\alpha$ ;
- два противоположных друг другу выражения устраняются, в результате чего создается пустое выражение; в этом случае из базы знаний  $KB$  следует высказывание  $\alpha$ .

**Листинг 7.5.** Простой алгоритм резолюции для пропозициональной логики. Функция **PL-Resolve** возвращает множество всех возможных выражений, полученных путем устранения противоположных друг другу литералов из двух высказываний, которые поступают на ее вход

---

```

function PL-Resolution(KB,  $\alpha$ ) returns значение true или false
    inputs: KB, база знаний – высказывание в пропозициональной логике
         $\alpha$ , запрос – высказывание в пропозициональной логике

    clauses  $\leftarrow$  множество выражений, полученное после преобразования
        высказывания KB  $\wedge \neg\alpha$  в форму CNF

    new  $\leftarrow$  {}
    loop do
        for each  $C_i, C_j$  in clauses do
            resolvents  $\leftarrow$  PL-Resolve( $C_i, C_j$ )
            if множество resolvents содержит пустое выражение
                then return true
            new  $\leftarrow$  new  $\cup$  resolvents
        if new  $\subseteq$  clauses then return false
        clauses  $\leftarrow$  clauses  $\cup$  new

```

---

Пустое выражение (дизъюнкция без дизъюнктов) эквивалентно высказыванию *False*, поскольку дизъюнкция является истинной, только если истинен по меньшей мере один из ее дизъюнктов. Еще один способ убедиться в том, что пустое выражение служит свидетельством противоречия, состоит в том, что, вернувшись к описанному выше процессу логического вывода, можно заметить, что пустое выражение возникает только после устранения двух взаимно противоположных единичных выражений, таких как  $P$  и  $\neg P$ .

Эту процедуру резолюции можно применить для формирования очень простого логического вывода в мире вампуса. Когда агент находится в квадрате  $[1, 1]$ , он не чувствует ветерка, поэтому в соседних квадратах не может быть ям. Соответствующая база знаний является следующей:

$$KB = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

и требуется доказать высказывание  $\alpha$ , которое, скажем, имеет вид  $\neg P_{1,2}$ . После преобразования высказывания ( $KB \wedge \neg \alpha$ ) в форму CNF получим выражения, показанные в верхнем ряду на рис. 7.6. В нижнем ряду на этом рисунке показаны все выражения, полученные путем устранения противоположных друг другу литералов из всех пар выражений, приведенных в верхнем ряду. Затем после устранения литерала  $P_{1,2}$ , противоположного литералу  $\neg P_{1,2}$ , будет получено пустое выражение, показанное в виде небольшого квадрата. Анализ результатов, приведенных на рис. 7.6, позволяет обнаружить, что многие этапы резолюции были бессмысленными. Например, выражение  $B_{1,1} \vee \neg B_{1,1} \vee P_{1,2}$  эквивалентно выражению *True*  $\vee P_{1,2}$ , которое эквивалентно *True*. Логический вывод, согласно которому выражение *True* является истинным, не очень полезен. Поэтому любое выражение, в котором присутствуют два взаимно дополнительных литерала, может быть отброшено.

### Полнота резолюции

Чтобы завершить обсуждение правила резолюции в этом разделе, покажем, почему алгоритм PL-Resolution является полным. Для этого целесообразно ввести

понятие **резолюционного замыкания**  $RC(S)$  множества выражений  $S$ , представляющего собой множество всех выражений, которые могут быть получены путем повторного применения правила резолюции к выражениям из множества  $S$  или к их производным. Резолюционным замыканием является множество выражений, которое вычисляется алгоритмом PL-Resolution в качестве окончательного значения переменной *clauses*. Можно легко показать, что множество  $RC(S)$  должно быть конечным, поскольку количество различных выражений, которые могут быть сформированы из символов  $P_1, \dots, P_k$ , присутствующих в  $S$ , является конечным. (Следует отметить, что это утверждение не было бы истинным, если бы не применялся этап факторизации, в котором уничтожаются дополнительные копии литералов.) Поэтому алгоритм PL-Resolution всегда оканчивает свою работу.

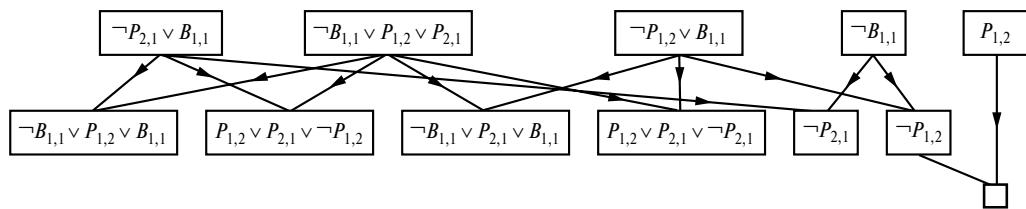


Рис. 7.6. Часть блок-схемы, показывающей процесс применения функции PL-Resolution для формирования простого логического вывода в мире атомов. Здесь показано, что из первых четырех выражений, приведенных в верхнем ряду, следует выражение  $\neg P_{1,2}$ .

Теорема полноты для правила резолюции в пропозициональной логике называется **основной теоремой резолюции**.

Если множество выражений является невыполнимым, то резолюционное замыкание этих выражений содержит пустое выражение.

Докажем эту теорему, показав, что справедливо противоположное ей утверждение: если замыкание  $RC(S)$  не содержит пустое выражение, то множество  $S$  выполнимо. В действительности для множества  $S$  можно создать модель с подходящими истинностными значениями для  $P_1, \dots, P_k$ . Процедура создания такой модели описана ниже.

Для  $i$  от 1 до  $k$ :

- если в множестве  $RC(S)$  имеется выражение, содержащее литерал  $\neg P_i$ , такое, что все другие его литералы являются ложными при данном присваивании, выбранном для  $P_1, \dots, P_{i-1}$ , то присвоить литералу  $P_i$  значение *false*;
- в противном случае присвоить литералу  $P_i$  значение *true*.

Остается показать, что это присваивание значений литералам  $P_1, \dots, P_k$  представляет собой модель множества выражений  $S$ , при условии, что множество  $RC(S)$  является замкнутым согласно правилу резолюции и не содержит пустого выражения. Доказательство этого утверждения оставляем читателю в качестве упражнения.

### Прямой и обратный логический вывод

Благодаря такой полноте алгоритм резолюции становится очень важным методом логического вывода. Однако во многих практических ситуациях вся мощь правила

резолюции не требуется. Реальные базы знаний часто содержат только выражения в ограниченной форме, называемые **хорновскими выражениями**. Хорновское выражение представляет собой дизъюнкцию литералов, среди которых положительным является не больше чем один. Например, выражение  $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ , где  $L_{1,1}$  означает, что местонахождением агента является квадрат [1, 1], представляет собой хорновское выражение, тогда как выражение  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$  таковым не является.

Ограничение, согласно которому только один литерал выражения должен быть положительным, может на первый взгляд показаться немного надуманным и беспersпективным, но фактически является очень важным по трем описанным ниже причинам.

1. Каждое хорновское выражение может быть записано как импликация, предпосылкой которой является конъюнкция положительных литералов, а заключением — один положительный литерал (см. упр. 7.12). Например, хорновское выражение  $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$  может быть записано как импликация  $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$ . В этой последней форме данное высказывание становится более легким для чтения: в нем утверждается, что если агент находится в квадрате [1, 1] и чувствует ветерок, то ветерок чувствуется в квадрате [1, 1]. Людям проще читать и писать в такой форме высказывания, касающиеся многих областей знаний.

Хорновские выражения, подобные этому, имеющие точно один положительный литерал, называются **определенными выражениями**. Такой положительный литерал называется **головой** выражения, а отрицательные литералы образуют **тело** выражения. Определенное выражение без отрицательных литералов просто утверждает справедливость некоторого высказывания; такую конструкцию иногда называют **фактом**. Определенные выражения образуют основу для **логического программирования**, которое рассматривается в главе 9. Хорновское выражение без положительных литералов может быть записано как импликация, заключением которой является литерал *False*. Например, выражение  $(\neg W_{1,1} \vee \neg W_{1,2})$ , согласно которому вампус не может находиться одновременно в квадратах [1, 1] и [1, 2], эквивалентно выражению  $W_{1,1} \wedge W_{1,2} \Rightarrow False$ . В мире баз данных такие высказывания называются **ограничениями целостности** и используются для обнаружения ошибок в данных. В приведенных ниже алгоритмах для простоты предполагается, что база знаний содержит только определенные выражения и не содержит ограничений целостности. Такие базы знаний мы будем называть *находящимися в хорновской форме*.

2. Логический вывод с использованием хорновских выражений может осуществляться с помощью алгоритма **прямого логического вывода** и **обратного логического вывода**, которые рассматриваются ниже. Оба эти алгоритма являются очень естественными в том смысле, что этапы логического вывода являются для людей очевидными и за ними можно легко проследить.
3. Получение логических следствий с помощью хорновских выражений может осуществляться за время, линейно зависящее от размера базы знаний.

Этот последний факт становится особенно приятным сюрпризом. Он означает, что процедура логического вывода оказывается весьма недорогостоящей применительно ко многим пропозициональным базам знаний, которые встречаются на практике.

Алгоритм прямого логического вывода  $\text{PL-FC-Entails?}(KB, q)$  определяет, следует ли единственный пропозициональный символ  $q$  (запрос) из базы знаний, представленной в форме хорновских выражений. Он начинает работу с известных фактов (положительных литералов) в базе знаний. Если известны все предпосылки некоторой импликации, то ее заключение добавляется к множеству известных фактов. Например, если известно, что имеют место факты  $L_{1,1}$  и  $Breeze$ , а в базе знаний имеется выражение  $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$ , то к ней можно добавить факт  $B_{1,1}$ . Этот процесс продолжается до тех пор, пока к базе знаний не добавляется запрос  $q$  или становится невозможным осуществление дальнейшего логического вывода. Этот подробный алгоритм приведен в листинге 7.6; главное, что следует о нем помнить, — то, что он действует за время, определяемое линейной зависимостью.

**Листинг 7.6. Алгоритм прямого логического вывода для пропозициональной логики.** В списке **agenda** (“повестка дня”) отслеживаются символы, в отношении которых известно, что они истинны, но которые еще не “обработаны”. В таблице **count** отслеживается информация о том, какое количество предпосылок каждой импликации еще не известно. Каждый раз, когда обрабатывается новый символ  $p$  из списка символов **agenda**, “стоящих на повестке дня”, количество предпосылок в таблице **count** сокращается на единицу для каждой импликации, в которой появляется предпосылка  $p$ . (Такие импликации могут обнаруживаться за постоянное время, если индексация базы знаний  $KB$  выполнена должным образом.) После того как количество неизвестных предпосылок для некоторой импликации достигает нуля, все эти предпосылки становятся известными, поэтому заключение этой импликации может быть добавлено к списку **agenda**. Наконец, необходимо следить за тем, какие символы уже были обработаны; символ, выведенный логическим путем, не следует добавлять к списку **agenda**, если он уже был обработан ранее. Это позволяет избежать излишней работы, а также предотвратить возникновение бесконечных циклов, которые могут быть вызваны наличием таких импликаций, как  $P \Rightarrow Q$  и  $Q \Rightarrow P$

---

```

function PL-FC-Entails?(KB, q) returns значение true или false
  inputs: KB, база знаний – множество пропозициональных
           хорновских выражений
           q, запрос – пропозициональный символ
  local variables: count, таблица, индексированная по высказываниям,
                     первоначально имеющая размер,
                     соответствующий количеству предпосылок
                     inferred, таблица, индексированная по символам,
                     в которой каждая запись первоначально
                     имеет значение false
                     agenda, список символов, первоначально включает
                     символы, известные как истинные
                     в базе знаний KB

  while список agenda не пуст do
    p  $\leftarrow$  Pop(agenda)
    unless inferred[p] do
      inferred[p]  $\leftarrow$  true
      for each хорновское выражение c в котором присутствует
                  предпосылка p do
        уменьшить на единицу значение count[c]
        if count[c] = 0 then do
          if Head[c] = q then return true
          Push(Head[c], agenda)
    return false

```

---

Лучший способ понять этот алгоритм состоит в том, чтобы рассмотреть пример и иллюстрацию. На рис. 7.7, а показана простая база знаний из хорновских выражений, содержащая выражения  $A$  и  $B$  как известные факты. На рис. 7.7, б приведена та же база знаний, изображенная в виде **графа AND-OR**. В графах AND-OR кратные дуги, соединенные кривой линией, обозначают конъюнкцию (в них необходимо доказать истинность каждой дуги), а кратные дуги, не соединенные друг с другом, обозначают дизъюнкцию (достаточно доказать истинность любой из этих дуг). На этом графе можно легко проверить, как действует алгоритм прямого логического вывода. Устанавливаются значения известных листовых узлов (в данном случае  $A$  и  $B$ ), а процесс логического вывода распространяется вверх по графу до тех пор, пока это возможно. При появлении любой конъюнкции процесс распространения останавливается и ожидает до тех пор, пока все конъюнкты не становятся известными, и только после этого продолжается дальше. Рекомендуем читателю подробно проработать этот пример.

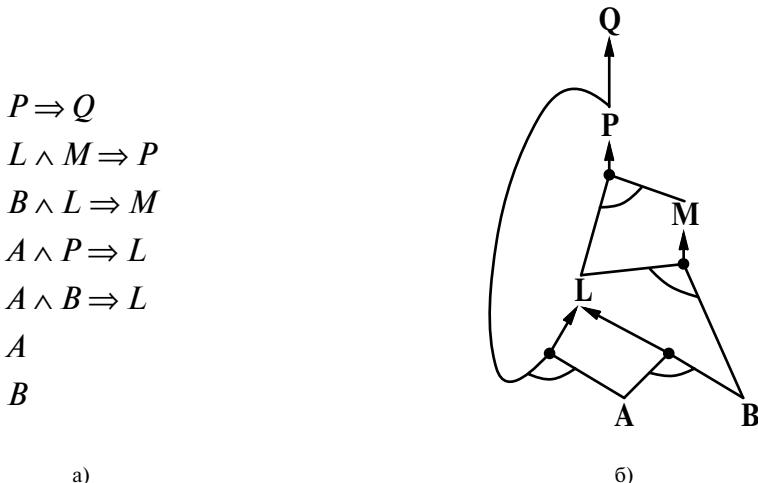


Рис. 7.7. Пример применения алгоритма прямого логического вывода: простая база знаний, состоящая из хорновских выражений (а); соответствующий граф AND-OR (б)

Можно легко убедиться в том, что алгоритм прямого логического вывода является **непротиворечивым**: каждый этап логического вывода по сути представляет собой применение правила отсечения. Кроме того, алгоритм прямого логического вывода является **полным**: в нем может быть получено каждое атомарное высказывание, которое следует из базы знаний. Проще всего в этом можно убедиться, рассмотрев заключительное состояние таблицы *inferred* (после того, как этот алгоритм достигает **фиксированной точки**, в которой становятся невозможными какие-либо новые этапы логического вывода). Эта таблица содержит значение *true* для каждого символа, выведенного логическим путем в течение этого процесса, и *false* — для всех других символов. Эта таблица может рассматриваться как логическая модель; более того, **каждое определенное выражение в первоначальной базе знаний КВ является истинным в данной модели**. Чтобы убедиться в этом, предположим обратное, а именно, что некоторое выражение  $a_1 \wedge \dots \wedge a_k \Rightarrow b$  из базы знаний является

ложным в этой модели. Тогда в этой модели выражение  $a_1 \wedge \dots \wedge a_k$  должно быть истинным и в ней же выражение  $b$  должно быть ложным. Но это противоречит нашему предположению о том, что алгоритм достиг фиксированной точки! Поэтому можно сделать вывод, что множество атомарных высказываний, полученное логическим путем к моменту достижения фиксированной точки, определяет модель первоначальной базы знаний  $KB$ . Более того, любое атомарное высказывание  $q$ , которое следует из базы знаний  $KB$ , должно быть истинным во всех ее моделях, а также, в частности, в данной модели. Итак, любое высказывание  $q$ , которое следует из базы знаний, должно быть выведено логическиенным алгоритмом.

Прямой логический вывод представляет собой пример применения общего понятия формирования логических рассуждений, управляемых данными, т.е. рассуждений, в которых фокус внимания вначале сосредоточен на известных данных. Прямой логический вывод может использоваться в любом агенте для получения заключений на основе поступающих результатов восприятия, часто даже без учета какого-либо конкретного запроса. Например, агент для мира вампира может сообщать с помощью операции  $\text{Tell}$  результаты своих восприятий базе знаний с использованием инкрементного алгоритма прямого логического вывода, в котором новые факты могут добавляться к “повестке дня” для инициализации новых процессов логического вывода. У людей формирование рассуждений, управляемых данными, в определенной степени происходит по мере поступления новой информации. Например, находясь дома и услышав, что пошел дождь, человек, который собирался на пикник, может решить, что придется его отменить. Но такое решение вряд ли будет принято, если он узнает, что оказался мокрым семнадцатый лепесток самой крупной розы в саду его соседа; люди держат процессы прямого логического вывода под тщательным контролем, поскольку в противном случае их просто затопил бы поток логических заключений, не имеющих ничего общего с реальностью.

Алгоритм обратного логического вывода, как указывает само его название, существует в обратном направлении от запроса. Если удается сразу же узнать, что высказывание, содержащееся в запросе  $q$ , является истинным, то не нужно выполнять никакой работы. В противном случае алгоритм находит те импликации в базе знаний, из которых следует  $q$ . Если можно доказать, что все предпосылки одной из этих импликаций являются истинными (с помощью обратного логического вывода), то высказывание  $q$  также является истинным. Будучи применен к запросу  $Q$ , показанному на рис. 7.7, этот алгоритм будет проходить вниз по графу до тех пор, пока не достигнет множества известных фактов, которые образуют основу для доказательства. Разработку подробного алгоритма оставляем читателю в качестве упражнения; как и в случае алгоритма прямого логического вывода, эффективная реализация этого алгоритма выполняет свою работу за линейное время.

Обратный логический вывод представляет собой одну из форм рассуждения, управляемого целями. Такая форма является полезной при получении ответов на конкретные вопросы, наподобие следующих: “Что теперь мне следует делать?” и “Где же находятся мои ключи?” Зачастую стоимость обратного логического вывода намного меньше по сравнению со стоимостью линейно зависящей от размера базы знаний, поскольку в этом процессе затрагиваются только факты, непосредственно относящиеся к делу. Вообще говоря, агент должен разделять работу между процессами прямого и обратного формирования рассуждений, ограничивая прямое формирование рассужде-

ний выработкой фактов, которые, по всей вероятности, будут относиться к запросам, подлежащим решению с помощью обратного логического вывода.

## 7.6. ЭФФЕКТИВНЫЙ ПРОПОЗИЦИОНАЛЬНЫЙ ЛОГИЧЕСКИЙ ВЫВОД

В данном разделе рассматриваются два семейства эффективных алгоритмов пропозиционального логического вывода, основанного на проверке по модели: один подход основан на поиске с возвратами, а другой — на поиске с восхождением к вершине. Указанные алгоритмы входят в состав основного “инструментария” пропозициональной логики. Этот раздел может быть пропущен при первом чтении настоящей главы.

Рассматриваемые здесь алгоритмы предназначены для проверки выполнимости. Выше уже отмечалась связь между поиском модели, обеспечивающей выполнимость логического высказывания, и поиском решения задачи удовлетворения ограничения, поэтому, скорее всего, нет ничего удивительного в том, что эти два семейства алгоритмов весьма напоминают алгоритмы поиска с возвратами, описанные в разделе 5.2, и алгоритмы локального поиска, которые представлены в разделе 5.3. Тем не менее приведенные здесь алгоритмы являются чрезвычайно важными сами по себе, поскольку в компьютерных науках существует очень много комбинаторных задач, которые можно свести к проверке выполнимости пропозиционального высказывания. Любое усовершенствование алгоритмов проверки выполнимости влечет за собой колоссальные последствия, связанные с повышением нашей способности справляться со всеми сложными задачами в целом.

### Полный алгоритм поиска с возвратами

Первый рассматриваемый здесь алгоритм часто называют *алгоритмом Дэвиса-Патнем* в честь авторов оригинальной статьи, в которой он был опубликован, Мартина Дэвиса и Хилари Патнем [336]. Затем этот алгоритм фактически стал одной из версий, описанных Дэвисом, Логеманом и Лавлендом [335], поэтому мы будем называть его DPLL по первым буквам фамилий всех четырех авторов. Алгоритм DPLL принимает на входе некоторое высказывание в конъюнктивной нормальной форме — высказывание, представленное как множество выражений. Как и алгоритмы Backtracking-Search и TT-Entails?, он фактически выполняет рекурсивный перебор в глубину всех возможных моделей. Но в этом алгоритме реализованы три описанных ниже усовершенствования, и этим он отличается от простой схемы, применяемой в алгоритме TT-Entails?.

- **Раннее завершение.** Алгоритм обнаруживает, должно ли данное высказывание быть истинным или ложным, даже с помощью частично завершенной модели. Выражение является истинным, если истинен любой его литерал, даже при том что для других литералов еще не определены истинностные значения; поэтому об истинности всего высказывания в целом можно судить еще до того, как модель будет составлена полностью. Например, высказывание  $(A \vee B) \wedge (A \vee C)$  является истинным, если истинен литерал  $A$ , незави-

сими от значений литералов  $B$  и  $C$ . Аналогичным образом, высказывание является ложным, если ложно любое его выражение, а это происходит, если каждый литерал этого выражения является ложным. Опять-таки, такая ситуация может возникнуть задолго до того, как модель будет полностью составлена. Раннее завершение позволяет обойтись без исследования целых поддеревьев в пространстве поиска.

- **Эвристика чистого символа.**  $\bowtie$  **Чистым символом** называется символ, который всегда появляется с одним и тем же “знаком” во всех выражениях. Например, в трех выражениях  $(A \vee \neg B)$ ,  $(\neg B \vee \neg C)$  и  $(C \vee A)$  символ  $A$  является чистым, поскольку он появляется только в виде положительных литералов, чистым можно также считать символ  $B$ , который появляется только в виде отрицательных литералов, а символ  $C$  считается нечистым. Можно легко показать, что если некоторое высказывание имеет модель, то это — модель с чистыми символами, значения которым присвоены так, чтобы их литералы приняли значение *true*, поскольку при таком условии ни одно выражение не может стать ложным. Следует отметить, что при определении чистоты символа алгоритм может игнорировать выражения, в отношении которых уже известно, что они истинны в модели, составленной до сих пор. Например, если модель содержит присваивание  $B=false$ , то выражение  $(\neg B \vee \neg C)$  уже является истинным, а символ  $C$  становится чистым, поскольку присутствует только в выражении  $(C \vee A)$ .
- **Эвристика единичного выражения.** **Единичное выражение** было определено ранее как выражение только с одним литералом. В контексте алгоритма DPLL оно также относится к выражениям, в которых в данной модели всем литералам, кроме одного, уже было присвоено значение *false*. Например, если модель содержит присваивание  $B=false$ , то выражение  $(B \vee \neg C)$  становится единичным выражением, поскольку оно эквивалентно выражению  $(False \vee \neg C)$ , или просто  $\neg C$ . Очевидно, для того, чтобы это выражение приняло истинное значение, литералу  $C$  должно быть присвоено значение *false*. Эвристика единичного выражения предусматривает присваивание значений всем таким символам до того, как происходит переход к обработке оставшейся части высказывания. Одним из важных следствий из этой эвристики является то, что любая попытка доказать (путем опровержения) истинность литерала, который уже находится в базе знаний, немедленно приводит к успеху (упр. 7.16). Следует также отметить, что присваивание значения одному единичному выражению может привести к созданию еще одного единичного выражения; например, после присваивания символу  $C$  значения *false* единичным становится выражение  $(C \vee A)$ , что влечет за собой присваивание истинного значения символу  $A$ . Такое “каскадное” распространение фиксированных присваиваний называется  $\bowtie$  **распространением единичных выражений**. Оно напоминает процесс прямого логического вывода с применением хорновских выражений, а в действительности, если рассматриваемое высказывание в конъюнктивной нормальной форме содержит только хорновские выражения, то алгоритм DPLL по сути сводится к алгоритму прямого логического вывода (см. упр. 7.17).

Алгоритм DPLL приведен в листинге 7.7. В этом листинге показана самая важная структурная часть алгоритма, которая описывает сам процесс поиска, но не представлены структуры данных, которые необходимо сопровождать для обеспечения эффективности каждого этапа поиска, а также исключены все программистские “хитрости”, которые можно было бы ввести для повышения производительности: изучение выражений, эвристики выбора переменных и операции рандомизированного перезапуска. После включения всех этих усовершенствований алгоритм DPLL, несмотря на свой почтенный возраст, становится одним из самых быстрых алгоритмов проверки выполнимости, которые когда-либо были разработаны. В частности, реализация Chaff этого алгоритма используется для решения задач проверки качества аппаратного обеспечения с миллионом переменных.

**Листинг 7.7. Алгоритм DPLL для проверки выполнимости высказывания в пропозициональной логике.** Принципы работы функций `Find-Pure-Symbol` и `Find-Unit-Clause` описаны в тексте; каждая из них возвращает символ (или неопределенное значение), а также истинностное значение, которое должно быть присвоено этому символу. Как и алгоритм `TT-Entails?`, этот алгоритм работает с частично составленными моделями

---

```

function DPLL-Satisfiable?(s) returns значение true или false
  inputs: s, высказывание в пропозициональной логике

  clauses  $\leftarrow$  множество выражений высказывания s, преобразованного
           в форму представления CNF
  symbols  $\leftarrow$  список пропозициональных символов в высказывании s
  return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns значение true или false

  if каждое выражение в множестве clauses имеет значение true
    в модели model
    then return true
  if какое-то выражение в множестве clauses имеет значение false
    в модели model
    then return false
  P, value  $\leftarrow$  Find-Pure-Symbol(symbols, clauses, model)
  if значение P не является пустым
    then return DPLL(clauses, symbols-P, Extend(P, value, model))
  P, value  $\leftarrow$  Find-Unit-Clause(clauses, model)
  if значение P не является пустым
    then return DPLL(clauses, symbols-P, Extend(P, value, model))
  P  $\leftarrow$  First(symbols); rest  $\leftarrow$  Rest(symbols)
  return DPLL(clauses, rest, Extend(P, true, model)) or
            DPLL(clauses, rest, Extend(P, false, model))

```

---

## Алгоритмы локального поиска

До сих пор в данной книге было представлено несколько алгоритмов локального поиска, включая алгоритмы Hill-Climbing (с. 176) и Simulated-Annealing (с. 181). Эти алгоритмы могут непосредственно применяться для решения задач проверки выполнимости, при условии, что будет выбрана правильная функция оценки. Поскольку цель состоит в том, чтобы найти присваивание, обеспечивающее выполне-

ние каждого выражения, для этой цели может использоваться любая функция оценки, которая подсчитывает количество невыполненных выражений. Фактически именно этот критерий и применяется в алгоритме Min-Conflicts для задач CSP (с. 227). Все эти алгоритмы делают шаги в пространстве полных присваиваний, каждый раз меняя на противоположное (инвертируя) истинностное значение одного символа. Это пространство обычно содержит много локальных минимумов, для выхода из которых требуются различные формы рандомизации. В последние годы было проведено множество экспериментов с тем, чтобы можно было найти приемлемый компромисс между стремлением к “жадному” выбору наилучшего преемника и необходимостью выходить из локальных минимумов с помощью случайного выбора очередного преемника.

Одним из простейших и наиболее эффективных алгоритмов, которые были созданы в результате всей этой работы, является алгоритм, получивший название WalkSAT (листинг 7.8). При каждой итерации этот алгоритм выбирает одно невыполненное выражение, а в этом выражении выбирает один символ для инвертирования. В данном алгоритме происходит случайным образом переключение между двумя способами выбора символа для инвертирования его истинностного значения: во-первых, этап с “минимизацией конфликтов”, в котором минимизируется количество невыполненных выражений в новом состоянии, и, во-вторых, этап “случайного передвижения”, в котором один из символов выбирается случайным образом.

**Листинг 7.8. Алгоритм WalkSAT для проверки выполнимости путем инвертирования значений переменных случайным образом. Существует много версий этого алгоритма**

---

```

function WalkSAT(clauses, p, max_flips) returns выполняющую
высказывание
    модель model или индикатор отказа failure
    inputs: clauses, множество выражений в пропозициональной логике
            p, вероятность выбора решения сделать ход со "случайным
            перемещением", которая, как правило, составляет около 0,5
    max_flips, количество инверсий, которые разрешено
            выполнить, прежде чем отказаться от дальнейших
            попыток

    model  $\leftarrow$  случайно выбранное присваивание значений true/false
            символам в выражениях
    for i = 1 to max_flips do
        if в модели model выполняется множество clauses
            then return model
        clause  $\leftarrow$  выбранное случайным образом выражение из множества
            clauses, которое имеет значение false в модели model
        with probability p инвертировать в модели model значение
            случайно выбранного символа из выражения clause
        else инвертировать значение того символа из выражения clause,
            который максимизирует количество выполненных выражений
            в множестве clauses
    return failure

```

---

Действительно ли такой алгоритм, как WalkSAT, способен выполнять производительную работу? Очевидно, что если он возвращает некоторую модель, то входное высказывание в самом деле выполнимо. А что если он возвращает индикатор неудачи *failure*? К сожалению, в этом случае невозможно определить, верно ли, что

высказывание является невыполнимым, или просто этому алгоритму требуется предоставить больше шансов добиться успеха. При этом можно попытаться присвоить параметру с определением максимального количества инверсий *max\_flips* бесконечно большое значение. В данном случае можно легко показать, что алгоритм WalkSAT в конечном итоге вернет какую-то модель (если она существует), при условии, что вероятность этого  $p > 0$ . Это связано с тем, что всегда существует последовательность инверсий, ведущая к присваиванию, которое обеспечивает выполнение высказывания, и такая последовательность в конечном итоге вырабатывается в результате случайного выбора этапов перемещения. К сожалению, если параметр *max\_flips* имеет бесконечно большое значение, а высказывание является невыполнимым, данный алгоритм никогда не заканчивает свою работу!

Из этого следует, что алгоритмы локального поиска, подобные WalkSAT, являются наиболее полезными, когда можно действительно рассчитывать на то, что решение существует; например, задачи, которые рассматривались в главах 3 и 5, обычно имеют решения. С другой стороны, локальный поиск не всегда может обнаружить невыполнимость, а именно это требуется, когда задача состоит в том, чтобы определить, следует ли какое-то высказывание из базы знаний. Например, агент не может надежно использовать локальный поиск для доказательства того, что некоторый квадрат в мире вампуса безопасен. Вместо этого он может лишь утверждать: “Я размышлял об этом в течение часа, но так и не мог найти хотя бы один из возможных миров, в котором данный квадрат не является безопасным”. А поскольку данный алгоритм локального поиска обычно позволяет действительно быстро найти модель, если она существует, то агент, использующий этот алгоритм, должен учитывать, что неудача в попытке найти модель высказывания с помощью данного алгоритма скорее всего свидетельствует о невыполнимости данного высказывания. Безусловно, такой результат — нечто иное, чем доказательство, и агент должен трижды подумать, прежде чем рискнуть своей жизнью, основываясь на подобных результатах.

### Трудные задачи определения выполнимости

Теперь рассмотрим, какие показатели производительности демонстрируют алгоритмы DPLL и WalkSAT на практике. Нас особенно интересуют трудные задачи, поскольку легкие могут быть решены с помощью любого старого алгоритма. В главе 5 мы ознакомились с некоторыми удивительными открытиями в области решения задач определенного типа. Например, задача с  $n$  ферзями (которая когда-то рассматривалась как чрезвычайно сложная) при ее решении с помощью алгоритмов поиска с возвратами оказалась тривиально простой для методов локального поиска, таких как метод с минимальными конфликтами. Это связано с тем, что решения этой задачи распределены в пространстве присваиваний очень плотно и для любого начального присваивания гарантируется, что решение находится недалеко. Таким образом, задача с  $n$  ферзями является простой потому, что она ~~недостаточно ограничена~~.

Если речь идет о решении задач проверки выполнимости, представленных в конъюнктивной нормальной форме, то среди них недостаточно ограниченными можно считать такие задачи, которые содержат относительно немного выражений, ограничиваю-

ших значения переменных. Например, ниже представлено выработанное случайным образом<sup>12</sup> высказывание в форме 3-CNF с пятью символами и пятью выражениями.

$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

Моделями этого высказывания являются 16 из 32 возможных вариантов присваивания, поэтому в среднем для поиска модели потребуются только две случайно выбранных гипотезы.

Так где же найти сложные задачи? Можно предположить, что если количество выражений будет увеличено, притом что количество символов останется постоянным, то задача станет более ограниченной и поиск решений окажется более затруднительным. Допустим, что  $m$  — количество выражений, а  $n$  — количество символов. На рис. 7.8, а показана вероятность того, что случайно сформированное высказывание в форме 3-CNF является выполнимым, как функция от отношения “выражение/символ”,  $m/n$ , при постоянном значении  $n$ , равном 50. Как и следовало ожидать, при малых значениях  $m/n$  эта вероятность близка к 1, а при больших значениях  $m/n$  вероятность близка к 0. На кривой вероятности начинается довольно резкий спад приблизительно после достижения значения  $m/n=4/3$ . Высказывания в форме CNF, приближающиеся к этой **критической точке**, можно назвать “почти выполнимыми”, или “почти невыполнимыми”. Можно ли считать, что именно здесь находятся трудные задачи?

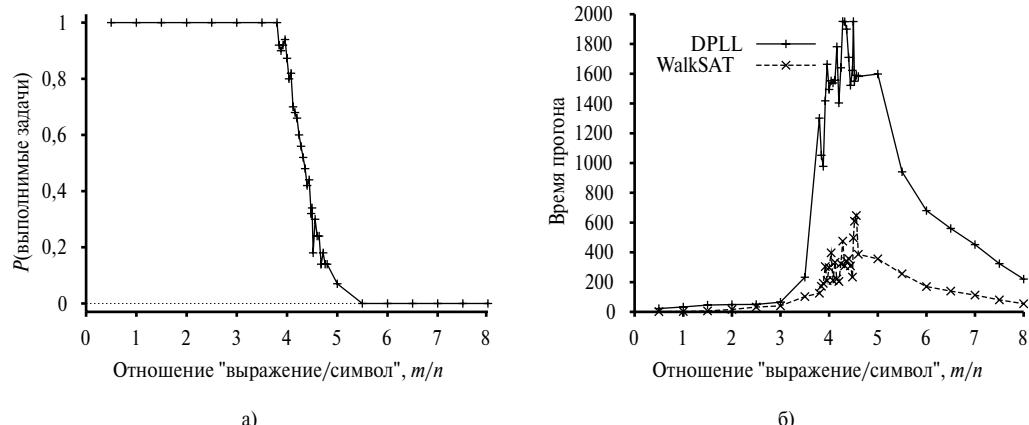


Рис. 7.8. Анализ производительности алгоритмов DPLL и WalkSAT при решении трудных задач: график, на котором показана вероятность того, что случайно сформированное высказывание в форме 3-CNF с количеством символов  $n=50$  окажется выполнимым, как функция от отношения “выражение/символ”,  $m/n$  (а); график усредненного времени прогона алгоритмов DPLL и WalkSAT применительно к 100 выполнимым сформированным случайнym образом высказываниям в форме 3-CNF с  $n=50$ , который показывает наличие узкого диапазона значений  $m/n$  вокруг критической точки (б).

На рис. 7.8, б показано время прогона алгоритмов DPLL и WalkSAT на участке вокруг этой критической точки, где наше внимание ограничивалось только выпол-

<sup>12</sup> Каждое выражение содержит три случайно выбранных различных символа, к каждому из которых применяется отрицание с вероятностью 50%.

нимыми задачами. Изучение приведенных результатов позволяет сделать три вывода: во-первых, задачи, приближающиеся к критической точке, являются гораздо более трудными по сравнению с другими случайно сформированными задачами; во-вторых, даже при решении самых сложных задач алгоритм DPLL является весьма эффективным — он требует выполнения в среднем нескольких тысяч шагов по сравнению со значением количества шагов  $2^{50} \approx 10^{15}$ , которое требуется для перебора истинностной таблицы; в-третьих, во всем этом диапазоне характеристик задач алгоритм WalkSAT работает намного быстрее, чем алгоритм DPLL.

Безусловно, эти результаты относятся только к случайно сформированным задачам. Реальные задачи не обязательно имеют такую же структуру, как случайно сформированные задачи (они могут характеризоваться разными значениями относительного количества положительных и отрицательных литералов, разными плотностями соединений между выражениями и т.д.). Тем не менее на практике алгоритм WalkSAT и подобные ему алгоритмы очень хорошо справляются также с решением реальных задач и часто не уступают самым лучшим алгоритмам специального назначения для этих задач. Такие решатели задач, как Chaff, легко справляются с задачами, состоящими из тысяч символов и миллионов выражений. На основании этих наблюдений можно сделать вывод, что некоторые комбинации эвристики с минимальными конфликтами и поведения со случайным блужданием предоставляют универсальную способность находить решения в большинстве ситуаций, в которых требуется комбинаторное формирование рассуждений.

## 7.7. АГЕНТЫ, ОСНОВАННЫЕ НА ПРОПОЗИЦИОНАЛЬНОЙ ЛОГИКЕ

В данном разделе соберем воедино все, что было описано до сих пор, для того, чтобы приступить к созданию агентов, действующих с использованием пропозициональной логики. Здесь будут рассматриваться два вида агентов: во-первых, агенты, в которых применяются алгоритмы логического вывода и база знаний, подобные показанному в листинге 7.1 универсальному агенту, основанному на знаниях, и, во-вторых, агенты, которые вычисляют значения логических высказываний непосредственно с помощью логических схем. В этом разделе будет также продемонстрировано функционирование агентов обоих типов в мире вампуша, а также показано, что оба они страдают от серьезных недостатков.

### Поиск ям и вампушов с помощью логического вывода

Начнем с описания агента, который рассуждает логически о том, где находятся ямы, вампусы и безопасные квадраты. Агент начинает свою работу с базы знаний, в которой описаны “законы” мира вампуша. Он знает, что квадрат  $[1, 1]$  не содержит яму или вампуша; это означает, что  $\neg P_{1,1}$  и  $\neg W_{1,1}$ . Для каждого квадрата  $[x, y]$  агенту известно высказывание с указанием того, как возникает ветерок:

$$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y}) \quad (7.1)$$

Для каждого квадрата  $[x, y]$  агенту известно высказывание с указанием того, как возникает неприятный запах:

$$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y}) \quad (7.2)$$

Наконец, он знает, что в мире вампуса существует точно один вампус. Соответствующее высказывание состоит из двух частей. Прежде всего необходимо указать, что имеется самое меньшее один вампус:

$$W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,3} \vee W_{4,4}$$

Затем необходимо указать, что существует самое большее один вампус. Один из способов формулировки этого утверждения состоит в том, что при наличии любых двух квадратов один из них обязательно должен быть свободным от вампуса. При наличии  $n$  квадратов мы получаем  $n(n-1)/2$  таких высказываний, которые подобны по форме высказыванию  $\neg W_{1,1} \vee \neg W_{1,2}$ . Таким образом, для мира с размерами  $4 \times 4$  описание начинается с общего количества 155 высказываний, содержащих 64 различных символа.

Программа агента, приведенная в листинге 7.9, сообщает в свою базу знаний с помощью операции `Tell` о каждом новом восприятии ветерка и неприятного запаха. (Она также обновляет некоторые обычные программные переменные для слежения за тем, где находится агент и где он побывал; дополнительная информация об этом приведена ниже.) Затем программа выбирает среди периферийных квадратов (т.е. квадратов, являющихся соседними по отношению к тем, которые уже посетил агент) такой квадрат, который должен быть проверен в следующую очередь. Агент может доказать, что периферийный квадрат  $[i, j]$  безопасен, если из базы знаний следует высказывание  $(\neg P_{i,j} \wedge \neg W_{i,j})$ . На втором месте по своей привлекательности находится квадрат, который, возможно, является безопасным; таковым является квадрат, применительно к которому агент не может доказать, что в нем имеется яма или вампус, т.е. квадрат, для которого из базы знаний не следует высказывание  $(P_{i,j} \vee W_{i,j})$ .

**Листинг 7.9.** Агент для мира вампуса, использующий пропозициональную логику для определения ям, вампусов и безопасных квадратов. Процедура `Route-Problem` формирует задачу поиска, решением которой является последовательность действий, ведущих от  $[x, y]$  до  $[i, j]$  и проходящих только через ранее посещенные квадраты

---

```

function PL-Wumpus-Agent(percept) returns действие action
  inputs: percept, список результатов восприятия в форме
            [Stench, Breeze, Glitter] ([ "Неприятный запах",
            "Ветерок", "Блеск"])
  static: KB, база знаний, первоначально содержащая лишь определения
            "законов функционирования" мира вампуса
            x, y, orientation, позиция агента (первоначально 1,1) и его
            ориентация (первоначально Right - смотрит
            вправо)
            visited, массив, показывающий, какие квадраты были посещены,
            первоначально содержащий значения false
            action, последнее по времени действие агента, первоначально
            не определено
            plan, намеченная последовательность действий, первоначально
            пустая

  обновить значения x, y, orientation, visited
  с учетом действия action
  if stench then Tell(KB, Sx,y) else Tell(KB,  $\neg S_{x,y}$ )
  if breeze then Tell(KB, Bx,y) else Tell(KB,  $\neg B_{x,y}$ )

```

```

if glitter then action  $\leftarrow$  grab
else if plan не пуст then action  $\leftarrow$  Pop(plan)
else if для некоторого периферийного квадрата  $[i, j]$ 
    результат Ask( $KB$ ,  $(\neg P_{i,j} \wedge \neg W_{i,j})$ ) имеет значение true or
    для некоторого периферийного квадрата  $[i, j]$  результат
    Ask( $KB$ ,  $(P_{i,j} \vee W_{i,j})$ ) имеет значение false then do
        plan  $\leftarrow$  A*-Graph-Search(Route-Problem( $[x, y]$ , orientation,
         $[i, j]$ , visited))
        action  $\leftarrow$  Pop(plan)
    else action  $\leftarrow$  случайно выбранный шаг
return action

```

Вычисление логического следствия в процедуре Ask может быть реализовано с использованием любого из методов, описанных выше в этой главе. Очевидно, что алгоритм TT-Entails? (см. листинг 7.3) является практически не применимым, поскольку в нем приходится выполнять перебор  $2^{64}$  строк. Алгоритм DPLL (см. листинг 7.7) осуществляет требуемый логический вывод за несколько миллисекунд в основном благодаря использованию эвристики с распространением единичных выражений. Может также использоваться алгоритм WalkSAT с учетом обычных предостережений о его неполноте. В рассматриваемых экземплярах мира вампуша неудачи в поиске модели при использовании 10 000 инверсий неизменно соответствуют невыполнимости, поэтому вероятность ошибок, вызванных неполнотой этого алгоритма, весьма мала.

В малом мире вампуша алгоритм PL-Wumpus-Agent действует весьма неплохо. Тем не менее некоторые особенности базы знаний агента остаются крайне неудовлетворительными. База знаний  $KB$  содержит “буквальные” высказывания в общей форме, приведенной в уравнениях 7.1 и 7.2, для каждого отдельного квадрата. Чем больше среда, тем крупнее должна быть такая начальная база знаний. Было бы гораздо лучше иметь только два высказывания, которые сообщают, из-за чего возникают ветерки и неприятные запахи в любых квадратах. Но такие выразительные возможности выходят за рамки пропозициональной логики. В следующей главе будет показан более выразительный логический язык, в котором можно легко представить подобные высказывания.

### Следение за местонахождением и ориентацией

Программа агента, приведенная в листинге 7.9, немного “жульничает”, поскольку в ней слежение за местонахождением осуществляется за пределами базы знаний, тогда как вместо этого следует использовать логические рассуждения<sup>13</sup>. Для выполнения этих функций “должным образом” потребуются высказывания, касающиеся местонахождения. Одна из первых попыток решения этой задачи может состоять в использовании некоторого символа наподобие  $L_{1,1}$  для обозначения того, что агент находится в квадрате  $[1, 1]$ . В таком случае начальная база знаний могла бы включать высказывания, подобные следующим:

<sup>13</sup> Наблюдательный читатель должен был заметить, что это позволило бы нам определить связь между чистыми восприятиями, такими как Breeze, и высказываниями, касающимися местонахождения, такими как  $B_{1,1}$ .

$$L_{1,1} \wedge FacingRight \wedge Forward \Rightarrow L_{2,1}$$

Однако сразу же обнаруживается, что такой способ является неприменимым. Если агент начинает движение с квадрата  $[1, 1]$ , глядя вправо, и передвигается вперед, то из базы знаний будут следовать высказывания и  $L_{1,1}$  (о его первоначальном местонахождении), и  $L_{1,2}$  (о его новом местонахождении). Тем не менее эти высказывания не могут одновременно быть истинными! Проблема состоит в том, что эти два высказывания, касающиеся местонахождения, должны ссылаться на два различных промежутка времени. Нам требуется ввести обозначение наподобие  $L_{1,1}^1$ , которое означало бы, что агент находился в квадрате  $[1, 1]$  в момент времени 1,  $L_{2,1}^2$  — для обозначения того, что агент находился в квадрате  $[2, 1]$  в момент времени 2, и т.д. Высказывания, касающиеся ориентации и действий, также должны зависеть от времени. Поэтому правильными высказываниями являются следующие:

$$\begin{aligned} L_{1,1}^1 \wedge FacingRight^1 \wedge Forward^1 &\Rightarrow L_{2,1}^2 \\ FacingRight \wedge TurnLeft^1 &\Rightarrow FacingUp^2 \end{aligned}$$

и тому подобные высказывания. Как оказалось, задача создания полной и правильной базы знаний, которая позволяла бы следить за всем, что происходит в мире вампуса, является весьма сложной; отложим полное обсуждение этой темы до главы 10. А в данном разделе достаточно лишь подчеркнуть такую мысль, что начальная база знаний должна содержать высказывания, подобные приведенным в двух предыдущих примерах, для каждого момента времени  $t$ , а также для каждого местонахождения. Это означает, что для каждого момента времени  $t$  и местонахождения  $[x, y]$  база знаний содержит некоторое высказывание примерно в такой форме:

$$L_{x,y}^t \wedge FacingRight^t \wedge Forward^t \Rightarrow L_{x+1,y}^{t+1} \quad (7.3)$$

Но даже если мы установим верхний предел количества допустимых интервалов времени (допустим, 100), в конечном итоге база знаний заполнится десятками тысяч высказываний. Та же самая проблема возникает, если для каждого нового интервала времени добавление высказываний происходит “по мере необходимости”. Из-за такого стремительного увеличения количества высказываний база знаний становится неудобной для чтения людьми, но быстрые пропозициональные алгоритмы — решатели задач все еще способны легко справляться с задачами в мире вампуса с размерами  $4 \times 4$  (они достигают предела своих возможностей в мире, размеры которого составляют приблизительно  $100 \times 100$ ). Агенты, основанные на логических схемах, которые описаны в следующем подразделе, предоставляют частичное решение этой проблемы стремительного увеличения количества высказываний, но нам придется подождать полного решения до того, как будет разработана логика первого порядка в главе 8.

## Агенты на основе логических схем

-Agent на основе логической схемы представляет собой особую разновидность рефлексных агентов с поддержкой состояния, которые определены в главе 2. В этом агенте результаты восприятия становятся входными данными для **последовательной логической схемы** — сети из **логических элементов**, каждый из которых реализует какую-то логическую связку, и **регистров**, каждый из которых хранит истинностное значение одного высказывания. Выходами этой логической схемы являются регистры, соответствующие действиям, например, выход *Grab* принимает значение *true*, если агент

хочет что-то схватить. Если вход *Glitter* (блеск) связан непосредственно с выходом *Grab*, то агент хватает свою цель, где бы он ее не обнаружил (рис. 7.9).

Вычисления значений с помощью логических схем осуществляются по принципу **обработки потока данных**: на каждом временном интервале устанавливаются значения входных данных и по логической схеме распространяются сигналы. Каждый раз, когда некоторый логический элемент получает на входе все необходимые данные, он вырабатывает на выходе некоторое значение. Этот процесс весьма напоминает процесс прямого логического вывода в таком графе AND-OR, какой приведен на рис. 7.7, б.

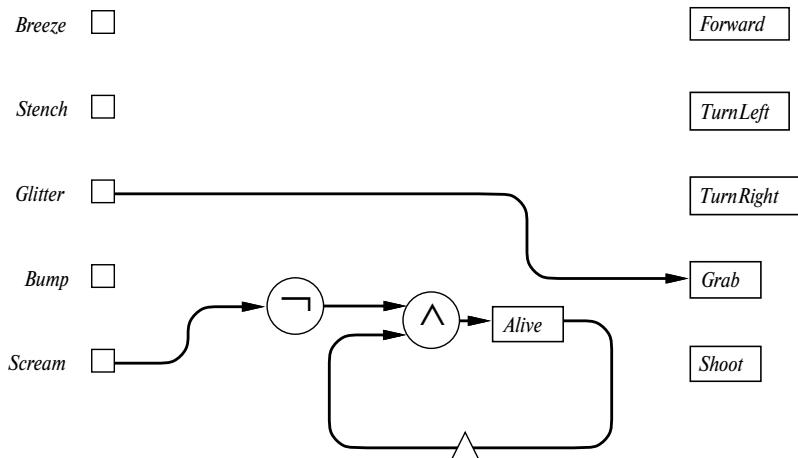


Рис. 7.9. Часть структурной схемы агента на основе логической схемы для мира вампира, на которой показаны входы, выходы, схема для захвата золота и схема для определения того, жив ли вампир. Регистры показаны в виде прямоугольников, а задержки на один шаг обозначаются небольшими треугольниками

В предыдущем разделе было указано, что агенты на основе логической схемы позволяют учитывать временные соотношения намного более успешно, чем пропозициональные агенты на основе логического вывода. Это связано с тем, что значение, хранящееся в каждом регистре, показывает истинностное значение соответствующего пропозиционального символа в текущий момент времени  $t$ , поэтому регистр не хранит отдельные копии этого значения для каждого отдельного интервала времени. Например, можно предусмотреть регистр *Alive*, который должен содержать значение *true*, если вампир жив, и *false*, если он мертв. Этот регистр соответствует пропозициональному символу  $\text{Alive}^t$ , поэтому в каждом временном интервале он ссылается на разные высказывания. Внутреннее состояние этого агента (т.е. его память) поддерживается путем обратного подключения выхода некоторого регистра к той же схеме через **линию задержки**. Это позволяет доставить в схему информацию о состоянии регистра в предыдущем интервале времени. Один из примеров такой конструкции приведен на рис. 7.9. Значение регистра *Alive* формируется с помощью конъюнкции отрицания восприятия *Scream* и пропущенного через линию задержки значения самого регистра *Alive*. Действие, выполняемое этой схемой применительно к регистру *Alive*, можно представить в форме высказывания как следующую двухстороннюю импликацию:

$$\text{Alive}^t \Leftrightarrow \neg \text{Scream}^t \wedge \text{Alive}^{t-1} \quad (7.4)$$

Это высказывание означает, что вампур в момент времени  $t$  жив тогда и только тогда, когда в момент времени  $t$  не был слышен его жалобный предсмертный крик (такое состояние нельзя определить по крику в момент времени  $t-1$ ), и он был жив в момент времени  $t-1$ . Предполагается, что при инициализации этой логической схемы значение регистра  $\text{Alive}$  устанавливается равным  $\text{true}$ . Поэтому значение регистра  $\text{Alive}$  остается истинным до тех пор, пока не раздается жалобный крик вампура, после чего это значение становится ложным и остается таковым. Это — именно то, что требуется.

За местонахождением агента можно следить в основном с использованием такого же способа, с помощью которого мы следим за здоровьем вампура. Для этого требуется иметь отдельный регистр  $L_{x,y}$  для каждого значения  $x$  и  $y$ ; его значение должно быть равным  $\text{true}$ , если агент находится в квадрате  $[x, y]$ . Однако логическая схема, которая присваивает значение регистру  $L_{x,y}$ , является гораздо более сложной по сравнению с логической схемой для регистра  $\text{Alive}$ . Например, агент находится в квадрате  $[1, 1]$  в момент времени  $t$ , если он, во-первых, был здесь в момент времени  $t-1$  и либо не двинулся вперед, либо попытался двинуться, но ударился об стену; во-вторых, находился в квадрате  $[1, 2]$ , смотря вниз, и двинулся вперед; в-третьих, находился в квадрате  $[2, 1]$ , смотря влево, и двинулся вперед. Эти три условия можно описать с помощью следующего высказывания:

$$\begin{aligned} L_{1,1}^t \Leftrightarrow & (L_{1,1}^{t-1} \wedge (\neg \text{Forward}^{t-1} \vee \text{Bump}^t)) \\ & \vee (L_{1,2}^{t-1} \wedge (\text{FacingDown}^{t-1} \wedge \text{Forward}^{t-1})) \\ & \vee (L_{2,1}^{t-1} \wedge (\text{FacingLeft}^{t-1} \wedge \text{Forward}^{t-1})) \end{aligned} \quad (7.5)$$

Схема для регистра  $L_{1,1}$  показана на рис. 7.10. К каждому регистру местонахождения подключена аналогичная схема. В упр. 7.13, б предлагается спроектировать логическую схему для высказываний, касающихся ориентации.

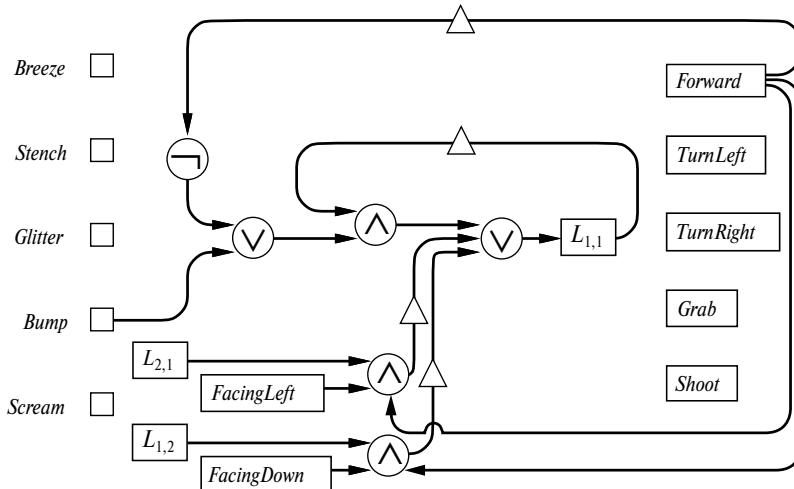


Рис. 7.10. Схема определения того, находится ли агент в квадрате  $[1, 1]$ . К каждому регистру с определением местонахождения и ориентации подключена аналогичная схема

Логические схемы, показанные на рис. 7.9 и 7.10, постоянно поддерживают правильные истинностные значения для регистров  $Alive$  и  $L_{x,y}$ . Однако необычным свойством логических высказываний, соответствующих этим схемам и регистрам, является то, что их правильные истинностные значения всегда можно проверить. С другой стороны, рассмотрим высказывание  $B_{4,4}$  о том, что в квадрате [4, 4] чувствуется ветерок. Несмотря на то что истинностное значение этого высказывания остается постоянным, агент не сможет узнать это истинностное значение до тех пор, пока не посетит квадрат [4, 4] (или не выведет логическим путем, что рядом с этим квадратом есть яма). Пропозициональная логика и логика первого порядка предназначены для того, чтобы с их помощью можно было автоматически представлять высказывания с истинными, ложными и неизвестными значениями, а логические схемы таким свойством не обладают: регистр для высказывания  $B_{4,4}$  обязан содержать хоть какое-то значение, либо *true*, либо *false*, даже несмотря на то, что истинные данные о том, каково его значение, еще не получены. Это означает, что значение в регистре вполне может оказаться неправильным, и это введет агента в заблуждение. Иными словами, в регистре должны быть представлены три возможных состояния (высказывание  $B_{4,4}$  является заведомо истинным, заведомо ложным или имеющим неизвестное значение), но в нашем распоряжении для этой цели имеется только один бит.

Решение такой проблемы состоит в использовании двух битов вместо одного. Высказывание  $B_{4,4}$  будет представлено с помощью двух регистров, которые мы будем называть  $K(B_{4,4})$  и  $K(\neg B_{4,4})$ , где  $K$  обозначает “known” (известный). (Напомним, что это все еще просто символы со сложными именами, даже несмотря на то, что они выглядят как структурированные выражения!) Если оба регистра,  $K(B_{4,4})$  и  $K(\neg B_{4,4})$ , содержат ложное значение, то истинностное значение  $B_{4,4}$  неизвестно. (А если оба они содержат истинное значение, то в базе знаний есть ошибка!) С этого времени, каждый раз, когда в некоторой части логической схемы нужно будет использовать высказывание  $B_{4,4}$ , вместо него будет применяться высказывание  $K(B_{4,4})$ , а когда потребуется использовать  $\neg B_{4,4}$ , вместо него будет служить  $K(\neg B_{4,4})$ . Вообще говоря, каждое потенциально неопределенное высказывание можно представить двумя **высказываниями с оценкой знаний** (knowledge proposition), которые позволяют определить, известно ли о том, что соответствующее высказывание является истинным, или известно, что оно ложно.

Примеры того, как использовать высказывания с оценкой знаний, будут вскоре приведены. Но вначале необходимо провести определенную работу, чтобы найти способ выяснения истинностных значений самих высказываний с оценкой знаний. Обратите внимание на то, что высказывание  $B_{4,4}$  имеет постоянное истинностное значение, а значения высказываний  $K(B_{4,4})$  и  $K(\neg B_{4,4})$  изменяются по мере того, как агент больше узнает об этом мире. Например, высказывание  $K(B_{4,4})$  вначале является ложным, а затем становится истинным, как только появляется возможность определить, что высказывание  $B_{4,4}$  является истинным, т.е. после того, как агент переходит в квадрат [4, 4] и обнаруживает ветерок. С тех пор оно продолжает оставаться истинным. Таким образом, справедливо следующее уравнение:

$$K(B_{4,4})^t \Leftrightarrow K(B_{4,4})^{t-1} \vee (L_{4,4}^t \wedge Breeze^t) \quad (7.6)$$

Аналогичное уравнение может быть составлено для высказывания  $K(\neg B_{4,4})^t$ .

Теперь, после того как агент узнал, в каких квадратах чувствуется ветерок, он может заняться обнаружением ям. В отсутствии ямы в некотором квадрате можно быть уве-

ренным тогда и только тогда, когда в отношении одного из соседних квадратов известно, что в нем не чувствуется ветерок. Например, имеет место следующее уравнение:

$$K(\neg P_{4,4})^t \Leftrightarrow K(\neg B_{3,4})^t \vee K(\neg B_{4,3})^t \quad (7.7)$$

Задача определения того, что в некотором квадрате есть яма, является более сложной — для этого в одном из соседних квадратов должен чувствоваться ветерок, который нельзя приписать наличию других ям, как показывает следующее уравнение:

$$\begin{aligned} K(P_{4,4})^t \Leftrightarrow & (K(B_{3,4})^t \wedge K(\neg P_{2,4})^t \wedge K(\neg P_{3,3})^t) \\ & \vee (K(B_{4,3})^t \wedge K(\neg P_{4,2})^t \wedge K(\neg P_{3,3})^t) \end{aligned} \quad (7.8)$$

Хотя логические схемы для определения наличия или отсутствия ям являются довольно сложными, ~~так~~ в них имеется лишь постоянное количество логических элементов в расчете на каждый квадрат. Это свойство является существенным, если мы занимаемся созданием агентов на основе логической схемы, предназначенных для решения задач, масштабы которых могут увеличиваться в разумных пределах. А фактически это — свойство самого мира вампуша; среда называется проявляющей свойство **локальности**, если истинность каждого интересующего нас высказывания можно определить, рассматривая лишь постоянное количество других высказываний. Обоснованность классификации по признаку локальности очень сильно зависит от того, насколько точно определена “физическая структура” данной среды. Например, область определения задачи игры в минного тральщика (Minesweeper) (упр. 7.11) является нелокальной, поскольку, чтобы узнать, есть ли мина в данном квадрате, может потребоваться проверить квадраты, находящиеся от него на произвольном расстоянии. Поэтому агенты на основе логических схем не всегда практически применимы для нелокальных областей определения.

Осталась еще одна проблема, которую мы до сих пор тщательно обходили, — вопрос об ~~и~~ **ацикличности**. Логическая схема является ацикличной, если каждый путь, соединяющий выход некоторого регистра в обратном направлении с его входом, содержит промежуточный элемент задержки. Мы требуем, чтобы все схемы были ацикличными, поскольку логические схемы, являющиеся цикличными, не работают как физические устройства! Они могут переходить в режим неустойчивых колебаний, что приводит к появлению неопределенных значений. В качестве примера цикличной схемы рассмотрим следующий дополненный вариант уравнения 7.6:

$$K(B_{4,4})^t \Leftrightarrow K(B_{4,4})^{t-1} \vee (L_{4,4}^t \wedge Breeze^t) \vee K(P_{3,4})^t \vee K(P_{4,3})^t \quad (7.9)$$

Дополнительные дизъюнкты,  $K(P_{3,4})^t$  и  $K(P_{4,3})^t$ , позволяют агенту определить, чувствуется ли ветерок, на основании известных данных о наличии ям в соседних квадратах, и, на первый взгляд, в этом нет ничего предосудительного. Но, к сожалению, появление ветерка зависит от наличия ям в соседних квадратах, а наличие ям зависит от того, чувствуется ли ветерок в соседних квадратах, и такая связь устанавливается с помощью уравнений, подобных уравнению 7.8. Поэтому полная логическая схема будет содержать циклы.

Сложность здесь заключается не в том, что дополненное уравнение 7.9 стало неправильным. Скорее всего, проблема состоит в том, что промежуточные зависимости, представленные подобными уравнениями, невозможно разрешить с помощью распространения истинностных значений по соответствующим логическим схемам. Ациклическая версия схемы, составленной с использованием уравнения 7.6, кото-

рая определяет наличие ветерка только с помощью прямого наблюдения, является неполной в том смысле, что могут возникать такие ситуации, когда агент на основе логической схемы окажется менее осведомленным, чем агент на основе логического вывода, использующий полную процедуру логического вывода. Например, если чувствуется ветерок в квадрате [1, 1], то агент на основе логического вывода может прийти к заключению, что ветерок чувствуется также в квадрате [2, 2], а агент на основе ациклической логической схемы, использующий уравнение 7.6, на это не способен. Задача создания полной логической схемы выполнима (в конце концов, логические схемы дают возможность эмулировать любой цифровой компьютер), но такая схема окажется значительно более сложной по сравнению с ациклической логической схемой.

### Сопоставление двух описанных типов агентов

Агент на основе логического вывода и агент на основе логической схемы представляют собой выражение двух противоположных подходов к проектированию агентов: декларативного и процедурного. Их сравнение может быть проведено по некоторым описанным ниже параметрам.

- **Краткость.** Агент на основе логической схемы, в отличие от агента на основе логического вывода, не обязан иметь отдельные копии своих “знаний”, относящиеся к каждому временному интервалу. Вместо этого он обращается только к данным, касающимся текущего и предыдущего временных интервалов. Для обоих агентов требуются копии описаний “физической структуры” (представленные в виде высказываний или логических схем) для каждого квадрата, и поэтому они не могут хорошо приспосабливаться к более крупным вариантам среды. В тех вариантах среды, характеризующихся наличием многочисленных объектов, между которыми установлены сложные связи, количество необходимых высказываний превосходит возможности любого пропозиционального агента. Для подобных вариантов среды требуется выразительная мощь логики первого порядка (см. главу 8). Кроме того, пропозициональные агенты обоих типов плохо приспособлены для представления или решения задачи поиска пути к ближайшему безопасному квадрату. (По этой причине в алгоритме PL-Wumpus-Agent приходится прибегать к использованию алгоритма поиска.)
- **Вычислительная эффективность.** В наихудшем случае логический вывод может потребовать времени, экспоненциально зависящего от количества символов, тогда как вычисление логического значения с помощью логической схемы требует времени, линейно зависящего от размера этой схемы (или линейно зависящего от глубины этой схемы, если она реализована в виде физического устройства). Однако на практике (как было показано выше) алгоритм DPLL выполняет требуемый логический вывод очень быстро<sup>14</sup>.

---

<sup>14</sup> Фактически все логические выводы, выполняемые с помощью логической схемы, могут быть сделаны с использованием алгоритма DPLL за линейное время! Это связано с тем, что в алгоритме DPLL вычисление по схеме (как и прямой логический вывод) может быть эмулировано с помощью правила распространения единичного выражения.

- **Полнота.** Выше было указано, что агент на основе логической схемы может быть неполным из-за ограничений, связанных с ацикличностью. Но фактически причины неполноты могут оказаться более фундаментальными. Прежде всего напомним, что продолжительность выполнения задания логической схемы линейно зависит от размера данной схемы. Это означает, что логическая схема для некоторых вариантов среды, являющаяся полной (т.е. позволяющая вычислить истинностное значение любого высказывания с определимым значением), должна быть экспоненциально больше, чем база знаний агента на основе логического вывода. В противном случае такая схема представляла бы собой реализацию некоего способа решения задачи поиска логического следствия в пропозициональной логике за время меньше экспоненциального, а это весьма маловероятно. Еще одна причина состоит в том, каков характер внутреннего состояния агента. Агент на основе логического вывода запоминает результаты каждого восприятия и обладает знаниями, либо явными, либо неявными, о каждом высказывании, которое следует из этих восприятий и начальной базы знаний. Например, получив результат восприятия  $B_{1,1}$ , этот агент знает о наличии дизъюнкции  $P_{1,2} \vee P_{2,1}$ , из которой следует высказывание  $B_{2,2}$ . Агент на основе логической схемы, с другой стороны, забывает все полученные ранее результаты восприятия и помнит только отдельные высказывания, хранящиеся в регистрах. Поэтому высказывания  $P_{1,2}$  и  $P_{2,1}$ , отдельно взятые, становятся для него неизвестными после получения результатов первого восприятия, т.е. он не может сделать вывод о том, что истинно высказывание  $B_{2,2}$ .
- **Простота конструирования.** Это — очень важный вопрос, на который нелегко найти точный ответ. Безусловно, авторы данной книги считают, что намного проще сформулировать высказывание о “физической структуре” декларативно, тогда как задача создания небольших, ациклических, не слишком неполных логических схем для непосредственного обнаружения ям представляется для них весьма сложной.

Подводя итог, можно прийти к выводу, что для согласования требований вычислительной эффективности, краткости, полноты и простоты конструирования необходимо прийти к определенным компромиссам. Если связь между восприятиями и действиями является простой (такой как связь между *Glitter* и *Grab*), оптимальным решением можно считать логическую схему, а для реализации более сложных связей может оказаться лучшим декларативный подход. Например, в такой проблемной области, как шахматы, декларативные правила являются краткими и простыми для кодирования (по крайней мере, в логике первого порядка), а логическая схема для вычисления ходов непосредственно по данным о позиции на доске была бы невообразимо огромной.

В царстве животных часто можно обнаружить различные проявления таких компромиссов. Низшие животные с очень простыми нервными системами, по-видимому, основаны на логических схемах, тогда как высшие животные, включая людей, очевидно, обладают способностью выполнять логический вывод на основе явных представлений. Это позволяет им вычислять гораздо более сложные функции агента. Люди имеют также логические схемы для реализации рефлексов, кроме того, возможно, обладают также способностью  **компилировать** декларатив-

ные представления для дальнейшего их использования в виде логических схем после того, как некоторые логические выводы становятся рутинными. Таким образом, проект **гибридного агента** (см. главу 2) может обладать лучшими способностями, взятыми из обоих миров.

## 7.8. РЕЗЮМЕ

В данной главе приведены вводные сведения об агентах на основе знаний, а также показано, как сформулировать логическое определение, с помощью которого такие агенты могут формировать рассуждения о мире, в котором они существуют. Основные идеи этой главы перечислены ниже.

- Интеллектуальным агентам требуются знания о мире для того, чтобы они могли вырабатывать хорошие решения.
- Знания содержатся в агентах в форме **высказываний на языке представления знаний**, которые хранятся в **базе знаний**.
- Агент на основе знаний состоит из базы знаний и механизма логического вывода. Он действует путем сохранения высказываний о мире в своей базе знаний, использования механизма логического вывода для получения новых высказываний и применения этих высказываний для принятия решения о том, какое действие следует выполнить.
- Язык представления определяется с помощью его **синтаксиса**, который задает структуру высказываний, и его **семантики**, которая определяет **истинность** каждого высказывания в каждом из **возможных миров**, или **модель** этого высказывания.
- Для понимания процесса формирования логических рассуждений крайне важно определить, как связаны **логические следствия** разных высказываний. Из высказывания  $\alpha$  следует другое высказывание  $\beta$ , если  $\beta$  является истинным во всех мирах, где истинно  $\alpha$ . Эквивалентные определения основаны на понятии **допустимости** высказывания  $\alpha \Rightarrow \beta$  и **невыполнимости** высказывания  $\alpha \wedge \neg\beta$ .
- Логический вывод — это процесс получения новых высказываний из старых. **Непротиворечивые** алгоритмы логического вывода обеспечивают получение только таких высказываний, которые являются логическими следствиями; **полные** алгоритмы обеспечивают получение всех высказываний, являющихся логическими следствиями.
- **Пропозициональная логика** — это очень простой язык, состоящий из **пропозициональных символов** и **логических связок**. Этот язык позволяет рассматривать высказывания, в отношении которых известно, являются ли они истинными, ложными или имеют полностью неизвестное логическое значение.
- При наличии постоянного словаря пропозициональных символов множество возможных моделей остается конечным, поэтому логическое следствие можно проверить, перебирая модели. Эффективные алгоритмы логического вывода на основе **проверки по модели** для пропозициональной логики включают мето-

ды поиска с возвратами и локального поиска и часто позволяют очень быстро решать крупные задачи.

- **Правила логического вывода** представляют собой шаблоны непротиворечивого логического вывода, которые могут использоваться для поиска доказательств. Правило **резолюции** позволяет создать полный алгоритм логического вывода для баз знаний, которые представлены в **конъюнктивной нормальной форме**. **Прямой логический вывод** и **обратный логический вывод** — это алгоритмы логического вывода, которые чрезвычайно естественно подходят для баз знаний, представленных в форме **хорновских выражений**.
- На основе пропозициональной логики могут быть созданы агенты двух типов: в **агентах на основе логического вывода** для слежения за миром и выявления его скрытых свойств используются алгоритмы логического вывода, тогда как в **агентах на основе логических схем** высказывания представлены в виде битов в регистрах, а обновление содержимого регистров происходит в результате распространения сигналов по логическим схемам.
- Пропозициональная логика является достаточно эффективной для решения некоторых задач в отдельно взятом агенте, но не позволяет распространить свое действие в масштабах таких вариантов среды, которые обладают неограниченными размерами, поскольку не имеет достаточной выразительной мощи, которая позволила бы кратко выразить временные, пространственные и другие универсальные шаблоны связей между объектами.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Понятие агента, который использует процесс формирования логических рассуждений для установления связи между восприятиями и действиями, было выдвинуто в статье Джона Маккарти “*Programs with Common Sense*” [1009], [1011]. Кроме того, Маккарти стал основоположником декларативного подхода, указав, что очень изящным является такой способ создания программного обеспечения, который предусматривает передачу агенту указаний о том, что он должен знать. В статье Аллена Ньюэлла “*The Knowledge Level*” [1124] подчеркнуто, что рациональные агенты могут быть описаны и проанализированы на абстрактном уровне, определяемом знаниями, которым они обладают, а не программами, которые в них выполняются. Сравнение декларативного и процедурного подходов к искусственноому интеллекту было проведено Боденом [145]. Незатухающие дебаты между приверженцами этих двух подходов снова ожили, после публикации [192] и [1145].

Сама логика имеет свои истоки в древнегреческой философии и математике. Фрагменты с изложением различных логических принципов (связывающих синтаксическую структуру высказываний с их истинным или ложным значением, с их смыслом или с допустимостью доводов, в которых они фигурируют) встречаются во многих трудах Платона. Первое известное систематическое исследование логики было проведено Аристотелем, работы которого были собраны его учениками после его смерти в 322 до н.э. в виде трактата, называемого *Органон*. ☰ **Силлогизмы** Аристотеля представляли собой то, что мы теперь называем *правилами логического вывода*. Хотя силлогизмы включали элементы и пропозициональной логики, и логики

первого порядка, сама система Аристотеля в целом была очень слабой с точки зрения современных стандартов. В ней не было места для шаблонов логического вывода, которые могли бы применяться к высказываниям с произвольной сложностью, как в современной пропозициональной логике.

Тесно связанные со школой Аристотеля мегарская и стоическая школы (которые зародились в пятом веке до н.э. и продолжали свою работу в течение нескольких столетий) ввели в научный обиход принципы систематического исследования импликации и других основных конструкций, до сих пор используемых в современной пропозициональной логике. Применение истинностных таблиц для определения логических связок было впервые предложено в трудах Филона из Мегары. Стоики приняли к использованию пять основных правил логического вывода как допустимые без доказательства, в том числе правило, которое теперь принято называть правилом отделения (*Modus Ponens*). Из этих пяти правил они вывели множество других правил, используя, кроме других принципов, теорему дедукции (с. 302), и имели гораздо более четкое представление о таком понятии, как доказательство, чем Аристотель. Стоики утверждали, что их логика была полной в том смысле, что они смогли описать все допустимые правила логического вывода, но от их трудов остались лишь отдельные фрагменты, по которым трудно судить об их правоте. Хорошее описание истории развития логики на примере мегарской и стоической школ, в той степени, в какой эти школы нам известны, приведено в работе Бенсона Мэйтса [1000].

Идеи о том, что логический вывод можно свести к чисто механическому процессу, применяемому к формальному языку, были высказаны Готфридом Вильгельмом Лейбницием (1646–1716). Однако собственные работы Лейбница в области математической логики обладали серьезными недостатками, поэтому он запомнился скорее как человек, выдвинувший эти идеи как цели, которые должны быть достигнуты, а не действительно предпринявший результивную попытку их достичь.

Джордж Буль [149] впервые представил полную и работоспособную систему формальной логики в своей книге *The Mathematical Analysis of Logic*. Логика Буля была почти полностью промоделирована на основе обычной алгебры действительных чисел, и в ней в качестве основного метода логического вывода использовалась подстановка логически эквивалентных выражений. Хотя систему Буля еще нельзя было считать полной пропозициональной логикой, она оказалась настолько близкой к такой, что другие математики сумели быстро заполнить все недостающие части. Шрёдер [1368] описал конъюнктивную нормальную форму, тогда как хорновская форма была введена намного позднее Альфредом Хорном [675]. Первое полное описание современной пропозициональной логики (и логики первого порядка) можно найти в книге *Begriffschrift* (“Система обозначения понятий”) Готтлоба Фреге [496].

Первое механическое устройство для формирования логических выводов было сконструировано потомственным графом Стенхуопом (1753–1816). Машина Стенхуопа, *Demonstrator*, была способна обрабатывать силлогизмы и некоторые логические выводы в теории вероятностей. Уильям Стэнли Джевонс, один из тех, кто усовершенствовал и расширил результаты Буля, сконструировал в 1869 году “логическое фортепиано” для формирования выводов в булевой логике. Занимательная и поучительная история этих и других ранних механических устройств для формирования рассуждений описана Мартином Гарднером [519]. Первой опубликованной компьютерной программой для формирования логического вывода была разработанная Ньюэллом, Шоу и Саймоном программа *Logic Theorist* [1127]. Эта

программа была предназначена для моделирования мыслительных процессов человека. Программа, позволяющая сформировать доказательство, была фактически спроектирована в 1954 году Мартином Дэвисом [334], но результаты работ в области создания программы Logic Theorist были опубликованы немного раньше. И программа Дэвиса от 1954 года, и программа Logic Theorist были основаны на достаточно произвольно выбранных методах, которые не оказали существенного влияния на дальнейшие работы в области автоматизированного дедуктивного вывода.

Истинностные таблицы как метод проверки допустимости или невыполнимости высказываний в языке пропозициональной логики были введены в арсенал ученых независимо Людвигом Виттгенштейном [1607] и Эмилем Постом [1231]. В 1930-х годах большие успехи были достигнуты в области создания методов логического вывода для логики первого порядка. В частности, Гёдель [565] показал, что полная процедура логического вывода в логике первого порядка может быть получена путем сведения к пропозициональной логике с использованием теоремы Эрбрана [650]. Мы снова вернемся к этой теме в главе 9, а здесь необходимо сделать важное замечание о том, что разработка эффективных пропозициональных алгоритмов в 1960-х годах мотивировалась в основном стремлением математиков создать эффективные средства доказательства теорем для логики первого порядка. Алгоритм Дэвиса-Патнем [336] был первым эффективным алгоритмом логического вывода в пропозициональной логике, но в большинстве случаев обладал меньшей эффективностью по сравнению с алгоритмом поиска с возвратами DPLL, который был введен двумя годами позже [335]. Полное правило резолюции и доказательство его полноты было опубликовано в оригинальной статье Дж.Э. Робинсона [1298], который также показал, как может осуществляться формирование рассуждений в логике первого порядка без обращения к пропозициональным методам.

Стефан Кук [289] показал, что задача определения выполнимости высказывания в пропозициональной логике является NP-полной. Поскольку определение того, является ли высказывание логическим следствием, эквивалентно задаче определения его невыполнимости, эта задача является ко-NP-полной. Известны многие подмножества пропозициональной логики, для которых задача проверки выполнимости решается за полиномиальное время; одним из таких подмножеств являются хорновские выражения. Алгоритм прямого логического вывода для хорновских выражений с линейными затратами времени предложен Доулингом и Галльером [407], которые описали свой алгоритм в виде процесса обработки потока данных, подобного распространению сигналов в логической схеме. Задача проверки выполнимости стала одним из канонических примеров сведения к NP-полным задачам; например, Кайе [782] показал, что игра Minesweeper (минный тральщик) (см. упр. 7.11) является NP-полной.

Попытки применения алгоритмов локального поиска для решения задач проверки выполнимости предпринимались различными авторами на протяжении всех 1980-х годов; все эти алгоритмы были основаны на идее минимизации количества невыполненных выражений [614]. Особенно эффективный алгоритм был разработан Гу [601] и независимо от него Селманом и др. [1382], которые назвали этот алгоритм GSAT и показали, что он способен решать широкий перечень очень трудных задач с очень большой скоростью. Алгоритм WalkSAT, описанный в этой главе, также предложен Селманом и др. [1380].

“Фазовый переход” в процессе определения выполнимости сформированных случайным образом задач k-SAT впервые был обнаружен Саймоном и Дюбуа [1421]. Эмпирические результаты, полученные Кроуфордом и Отоном [307], свидетельствуют о том, что при решении крупных сформированных случайным образом задач 3-SAT это резкое изменение характеристик возникает в диапазоне значений отношения “высказывание/переменная”, приближающихся к 4,24; кроме того, в этой статье описана очень эффективная реализация алгоритма DPLL. Байардо и Шрэг [87] описали еще одну эффективную реализацию алгоритма DPLL с использованием методов из области удовлетворения ограничений, а в [1090] описан алгоритм Chaff, который решает задачи проверки аппаратного обеспечения с миллионом переменных, ставший победителем конкурса SAT 2002. Ли и Энбулаган [926] описали эвристики, основанные на распространении единичных выражений, позволяющие создавать быстрые решатели задач. Чизман и др. [244] предоставили данные о многих задачах, связанных с задачами проверки выполнимости, и пришли к выводу, что все NP-трудные задачи обнаруживают фазовый переход. Кёркпатрик и Селман [800] показали, каким образом можно использовать методы статистической физики для получения представления о том, какова точная “форма” фазового перехода. Теоретический анализ его местонахождения все еще довольно неудовлетворителен: до сих пор удалось доказать лишь то, что для случайно сформированных задач 3-SAT фазовый переход находится в диапазоне [3.003, 4.598]. Кук и Митчелл [290] составили превосходный обзор результатов по этой и некоторым другим темам, касающимся выполнимости.

Самые ранние теоретические исследования показали, что алгоритм DPLL применительно к некоторым естественным распределениям задач характеризуется в среднем полиномиальной сложностью. Этот факт, который мог бы, в принципе, стать предметом восхищения, начал казаться менее восхитительным, когда Франко и Паулл [494] показали, что те же задачи можно было бы решить за постоянное время, просто проверяя случайно выбранные варианты присваивания. Метод случайной выработки вариантов, описанный в данной главе, приводит к получению гораздо более трудных задач. Заинтересовавшись достигнутым на практике успехом в использовании локального поиска при решении таких задач, Кутсупиас и Пападимитриу [848] показали, что простой алгоритм восхождения к вершине способен решать почти все экземпляры задачи проверки выполнимости очень быстро, а это свидетельствует о том, что трудные задачи встречаются редко. Более того, Шёнинг [1365] продемонстрировал рандомизированный вариант алгоритма GSAT, для которого ожидаемое время прогона в худшем случае при решении задач 3-SAT составляет  $1.333^n$ ; этот предел все еще является экспоненциальным, но гораздо более низким по сравнению с достигнутыми ранее пределами для худшего случая. Алгоритмы проверки выполнимости все еще остаются очень активной областью исследования; хорошей отправной точкой для изучения этой темы может служить сборник статей под редакцией Дю и др. [418].

Истоки идей по созданию агентов на основе логических схем можно проследить до оригинальной статьи Мак-Каллюка и Питтса [1017], которая послужила началом исследований в области нейронных сетей. Вопреки сложившемуся общему мнению, эта статья касалась реализации проекта агента на основе логической схемы в мозгу. Однако агенты на основе логической схемы долго не привлекали значительного внимания

исследователей в области искусственного интеллекта. Самым заметным исключением из этого правила явились работы Стена Розеншайна [760], [1308], который разработал способы компиляции агентов на основе логических схем из декларативных описаний проблемной среды. Логические схемы для обновления высказываний, хранящихся в регистрах, тесно связаны с **аксиомой состояния-преемника**, разработанной для логики первого порядка Рейтером [1277]. В работах Рода Брукса [189], [190] продемонстрирована эффективность использования проектов на основе логической схемы для управления роботами; к этой теме мы вернемся в главе 25. Брукс [192] утверждает, что для искусственного интеллекта необходимы лишь проекты на основе логических схем, а методы представления и формирования логических рассуждений являются громоздкими, дорогостоящими и ненужными. Но, по мнению авторов, ни один из этих подходов, отдельно взятых, не является достаточным.

Мир вампуса был придуман Грегори Йобом [1633]. Любопытно то, что Йоб разработал этот мир, поскольку ему надоели игры, которые проводятся в мире, представленном в виде квадратной решетки: топология его первоначального мира вампуса представляла собой додекаэдр, а мы снова поместили вампуса в старую скучную квадратную решетку. Майкл Генезерет был первым, кто указал, что мир вампуса можно использовать как испытательную площадку для агентов.

## УПРАЖНЕНИЯ

- 7.1. Опишите мир вампуса в соответствии со свойствами проблемной среды, перечисленными в главе 2.
- 7.2. Предположим, что агент достиг пункта, показанного на рис. 7.3, *a*, получив такие результаты восприятия: в квадрате  $[1, 1]$  — ничего, в квадрате  $[2, 1]$  — ветерок, а в квадрате  $[1, 2]$  — неприятный запах. В настоящий момент агента интересует содержимое квадратов  $[1, 3]$ ,  $[2, 2]$  и  $[3, 1]$ . Каждый из них может содержать яму, и самое большое в одном из них может находиться вампус. На основе примера, приведенного на рис. 7.4, сконструируйте множество возможных миров. (Должно быть найдено 32 таких возможных мира.) Отметьте миры, в которых существующая база знаний является истинной, а также те, в которых истинным является каждое из следующих высказываний:

$\alpha_2 = \text{"В квадрате } [2, 2] \text{ нет ямы"}$

$\alpha_3 = \text{"В квадрате } [1, 3] \text{ есть вампус"}$

На основании этого покажите, что  $KB \models \alpha_2$  и  $KB \models \alpha_3$ .

- 7.3. Рассмотрите задачу определения того, является ли некоторое высказывание пропозициональной логики истинным в данной модели.
  - a) Напишите рекурсивный алгоритм  $PL\text{-True?}(s, m)$ , который возвращает  $true$  тогда и только тогда, когда высказывание  $s$  является истинным в модели  $m$  (где  $m$  присваивает истинностное значение каждому символу в  $s$ ). Этот алгоритм должен заканчивать свою работу за время, линейно зависящее от размера высказывания. (Еще один вариант состоит в том, чтобы воспользоваться версией этой функции из оперативного репозитария кода.)

- 6)** Приведите три примера высказываний, в отношении которых можно определить, являются ли они истинными или ложными, в частичной модели, в которой не заданы истинностные значения для некоторых из символов.
- в)** Покажите, что в общем случае истинностное значение высказывания (если оно имеется) невозможно эффективно определить в частичной модели.
- г)** Доработайте свой алгоритм PL-True? так, чтобы он иногда позволял судить об истинностном значении по частичным моделям, сохраняя вместе с тем свою рекурсивную структуру и линейную зависимость времени прогона от размера высказывания. Приведите три примера высказываний, истинность которых в частичной модели не обнаруживается вашим алгоритмом.
- д)** Проведите исследование того, позволяет ли этот модифицированный алгоритм повысить эффективность алгоритма TT-Entails?.
- 7.4.** Докажите каждое из приведенных ниже утверждений.
- Высказывание  $\alpha$  является допустимым тогда и только тогда, когда  $\text{True} \models \alpha$ .
  - Для любого высказывания  $\alpha$  истинно, что  $\text{False} \models \alpha$ .
  - Истинно, что  $\alpha \models \beta$  тогда и только тогда, когда высказывание  $(\alpha \Rightarrow \beta)$  допустимо.
  - Истинно, что  $\alpha \equiv \beta$  тогда и только тогда, когда высказывание  $(\alpha \Leftrightarrow \beta)$  допустимо.
  - Истинно, что  $\alpha \models \beta$  тогда и только тогда, когда высказывание  $(\alpha \wedge \neg \beta)$  недостижимо.
- 7.5.** Рассмотрите словарь, состоящий только из четырех высказываний,  $A$ ,  $B$ ,  $C$  и  $D$ . Сколько существует моделей для каждого из следующих высказываний?
- $(A \wedge B) \vee (B \wedge C)$
  - $A \vee B$
  - $A \Leftrightarrow B \Leftrightarrow C$
- 7.6.** В этой главе мы определили четыре различных бинарных логических связки.
- Имеются ли какие-либо другие логические связки, которые могут оказаться полезными?
  - Сколько может быть определено разных бинарных связок?
  - Почему некоторые из них не очень полезны?
- 7.7.** Используя любой предпочтаемый вами метод, проверьте каждую из эквивалентностей, приведенных в листинге 7.4.
- 7.8.** Определите, является ли каждое из приведенных ниже высказываний действительным, невыполнимым или ни тем ни другим. Проверьте полученные вами результаты с помощью истинностных таблиц или правил эквивалентности, приведенных в листинге 7.4.
- $\text{Smoke} \Rightarrow \text{Smoke}$
  - $\text{Smoke} \Rightarrow \text{Fire}$
  - $(\text{Smoke} \Rightarrow \text{Fire}) \Rightarrow (\neg \text{Smoke} \Rightarrow \neg \text{Fire})$

- и)  $\text{Smoke} \vee \text{Fire} \vee \neg\text{Fire}$
- д)  $((\text{Smoke} \wedge \text{Heat}) \Rightarrow \text{Fire}) \Leftrightarrow ((\text{Smoke} \Rightarrow \text{Fire}) \vee (\text{Heat} \Rightarrow \text{Fire}))$
- е)  $(\text{Smoke} \Rightarrow \text{Fire}) \Rightarrow ((\text{Smoke} \wedge \text{Heat}) \Rightarrow \text{Fire})$
- ж)  $\text{Big} \vee \text{Dumb} \vee (\text{Big} \Rightarrow \text{Dumb})$
- з)  $(\text{Big} \wedge \text{Dumb}) \vee \neg\text{Dumb}$
- 7.9. (Адаптировано из [78].) Можете ли вы доказать на основании приведенных ниже рассуждений, что единорог — мифическое существо? А что насчет того, что это волшебное существо? Существо, вооруженное рогом?
- Если единорог — мифическое существо, то он бессмертен, а если не мифическое, то он — смертное млекопитающее. Если единорог либо бессмертен, либо является млекопитающим, то он вооружен рогом. Единорог является волшебным существом, если он вооружен рогом.
- 7.10. Любое высказывание пропозициональной логики логически эквивалентно утверждению, что любой возможный мир, в котором это высказывание было бы ложным, не имеет места. На основании этого наблюдения докажите, что любое высказывание может быть записано в конъюнктивной нормальной форме.
- 7.11. Minesweeper (минный тральщик) — широко известная компьютерная игра, которая тесно связана с миром вампира. Мир минного тральщика представляет собой прямоугольную решетку из  $N$  квадратов с  $M$  разбросанными по ним невидимыми минами. Агент может проверить любой квадрат; если он проверяет заминированный квадрат, его ждет немедленная смерть. В игре Minesweeper наличие мин раскрывается путем показа в каждом проверенном квадрате количества мин в квадратах, соседних по горизонтали, вертикали или диагонали. Цель игры состоит в том, чтобы проверить каждый незаминированный квадрат.
- а) Допустим, что высказывание  $X_{i,j}$  является истинным тогда и только тогда, когда в квадрате  $[i, j]$  находится мина. Составьте утверждение, согласно которому в квадратах, соседних по отношению к квадрату  $[1, 1]$ , имеется точно две мины, в виде высказывания, включающего определенную логическую комбинацию высказываний  $X_{i,j}$ .
- б) Обобщите ваше утверждение, составленное при выполнении упр. 7.11, а, и объясните, как следует формировать высказывание CNF с утверждением, что  $k$  из  $n$  соседних квадратов содержат мины.
- в) Точно объясните, как агент может использовать алгоритм DPLL для доказательства того, что данный квадрат содержит (или не содержит мину), игнорируя глобальное ограничение, согласно которому всего имеется точно  $M$  мин.
- г) Предположим, что глобальное ограничение конструируется с помощью метода, созданного вами при выполнении упр. 7.11, б. Как зависит количество выражений в этом ограничении от  $M$  и  $N$ ? Предложите способ осуществления такой модификации алгоритма DPLL, чтобы в нем не требовалось явно представлять это глобальное ограничение.

- д) Становятся ли недействительными какие-либо выводы, полученные при создании метода, о котором речь идет в задании 7.11, *в*, если учитывается это глобальное ограничение?
- е) Приведите примеры конфигураций проверочных значений, которые порождают такие далеко идущие зависимости, что содержимое любого не-проверенного квадрата может предоставить информацию о содержимом какого-то далеко отстоящего от него квадрата. (*Подсказка.* Рассмотрите доску с размерами  $N \times 1$ .)
- 7.12.** В данном упражнении рассматривается связь между выражениями и импликативными высказываниями.
- Покажите, что выражение  $(\neg P_1 \vee \dots \vee \neg P_m \vee Q)$  логически эквивалентно импликативному высказыванию  $(P_1 \wedge \dots \wedge P_m) \Rightarrow Q$ .
  - Покажите, что каждое выражение (независимо от количества положительных литералов) может быть записано в форме  $(P_1 \wedge \dots \wedge P_n) \Rightarrow (Q_1 \vee \dots \vee Q_n)$ , где  $P$  и  $Q$  — пропозициональные символы. База знаний, состоящая из таких высказываний, находится в **импликативной нормальной форме**, или в **форме Ковалевского**.
  - Составьте полное правило резолюции для высказываний в импликативной нормальной форме.
- 7.13.** В этом упражнении речь идет о проектировании агента для мира вампуса на основе логической схемы.
- Запишите уравнение, аналогичное уравнению 7.4, для высказывания *Arrow*, которое должно быть истинным, если у агента все еще есть стрела. Составьте соответствующую логическую схему.
  - Повторите задание 7.13, *а* для высказывания *FacingRight* (агент смотрит вправо), используя в качестве образца уравнение 7.5.
  - Создайте версии уравнений 7.7 и 7.8 для поиска вампуса и начертите логическую схему.
- 7.14.** Обсудите понятие оптимального поведения в мире вампуса. Покажите, что приведенное в этой главе определение алгоритма PL-Wumpus-Agent не является оптимальным, и предложите способы его усовершенствования.
- 7.15.** Дополните алгоритм PL-Wumpus-Agent таким образом, чтобы он обеспечивал отслеживание всех относящихся к делу фактов с помощью только базы знаний.
- 7.16.** Сколько времени потребуется, чтобы доказать высказывание  $KB \models \alpha$  с помощью алгоритма DPLL, если  $\alpha$  — литерал, который уже содержится в базе знаний  $KB$ ? Объясните полученные результаты.
- 7.17.** Проследите за поведением алгоритма DPLL при обработке приведенной на рис. 7.7 базы знаний в попытке доказать высказывание  $Q$  и сравните это поведение с тем, которое демонстрируется алгоритмом прямого логического вывода.

# 8 ЛОГИКА ПЕРВОГО ПОРЯДКА

*В данной главе мы обращаем внимание на то, что мир состоит из множества объектов, причем некоторые из них связаны с другими объектами, а также предпринимаем попытку рассуждать об этих объектах.*

В главе 7 было показано, каким образом агент, основанный на знаниях, может представлять мир, в котором он действует, и определять с помощью логического вывода, какие действия следует ему предпринять. В этой главе в качестве языка представления использовалась пропозициональная логика, которая вполне позволяет проиллюстрировать основные понятия логики и принципы функционирования агентов, основанных на знаниях. Но, к сожалению, язык пропозициональной логики слишком слаб для того, чтобы с его помощью можно было кратко представить знания о сложных вариантах среды. В настоящей главе рассматривается **логика первого порядка**<sup>1</sup>, которая является достаточно выразительной для того, чтобы с ее помощью можно было представить значительную часть наших общих знаний. Кроме того, логика первого порядка либо становится итогом развития, либо образует основу многих других языков представления, и ей посвящены многие десятилетия интенсивных исследований. Начнем изложение этой темы с описания в разделе 8.1 всех языков представления в целом; в разделе 8.2 рассматриваются синтаксис и семантика логики первого порядка; в разделах 8.3 и 8.4 иллюстрируется использование логики первого порядка для простых представлений.

## 8.1. ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О ПРЕДСТАВЛЕНИИ

В данном разделе рассматривается общий характер языков представления. Здесь будет показано, чем была вызвана необходимость разработки логики первого порядка — гораздо более выразительного языка по сравнению с пропозициональной логикой, представленной в главе 7. Мы рассмотрим язык пропозициональной логики

<sup>1</sup> Логику первого порядка называют также **исчислением предикатов первого порядка**, и эти области знаний иногда сокращенно обозначают как **FOL** — First-Order Logic или **FOPC** — First-Order Predicate Calculus.

и языки других типов, чтобы понять, в чем состоят их преимущества и недостатки. Приведенное здесь описание будет кратким, и в нем результаты многовековых размышлений, проб и ошибок сведутся всего лишь к нескольким абзацам.

Среди наиболее широко распространенных формальных языков наиболее крупным классом являются языки программирования (такие как C++, Java или Lisp). Сами программы представляют, в прямом смысле этого понятия, только вычислительные процессы, а структуры данных, применяемые в программах, могут представлять факты; например, в программе для представления содержимого мира вампуса может использоваться массив  $4 \times 4$ . Поэтому оператор  $World[2, 2] \leftarrow Pit$  языка программирования представляет собой довольно естественный способ формирования утверждения о том, что в квадрате  $[2, 2]$  имеется яма. (Такие способы представления могут рассматриваться как выбранные произвольным образом; системы баз данных были разработаны именно для предоставления более общего способа хранения и выборки фактов, независимого от проблемной области.) В языках программирования недостает какого-то общего механизма логического вывода фактов на основе других фактов; каждое обновление в структуре данных осуществляется с помощью процедуры, характерной для данной проблемной области, подробности устройства которой уточняются программистом на основании его знаний о проблемной области. Такой **процедурный** подход представляет собой резкий контраст с **декларативным** характером пропозициональной логики, в которой не смешиваются знания и методы логического вывода, а логический вывод осуществляется полностью независимо от проблемной области.

Еще одним недостатком применения структур данных в программах (а также, в этом отношении, и баз данных) является отсутствие удобного способа сформировать, например, такое утверждение: “В квадрате в  $[2, 2]$  или  $[3, 1]$  имеется яма” или “Если вампус находится в квадрате  $[1, 1]$ , то его нет в квадрате  $[2, 2]$ ”. Программы позволяют хранить для каждой переменной единственное значение, а некоторые системы допускают использование “неопределенных” значений, но в них не хватает выразительности, которая требуется для обработки частичной информации.

Пропозициональная логика — это декларативный язык, поскольку ее семантика основана на истинностных отношениях между высказываниями и возможными мирами. Кроме того, она имеет достаточную выразительную мощь для того, чтобы с ее помощью можно было обрабатывать частично заданную информацию с использованием дизъюнкции и отрицания. Пропозициональная логика обладает еще одним свойством, которое является желательным для языков представления, а именно **композиционностью**. В композициональном языке смысл высказывания представляет собой функцию от смысла его частей. Например, смысл высказывания “ $S_{1,4} \wedge S_{1,2}$ ” связан со смыслами высказываний “ $S_{1,4}$ ” и “ $S_{1,2}$ ”. Было бы очень странным, если бы “ $S_{1,4}$ ” означало, что в квадрате  $[1, 4]$  чувствуется неприятный запах, “ $S_{1,2}$ ” — что неприятный запах чувствуется в квадрате  $[1, 2]$ , а выражение “ $S_{1,4} \wedge S_{1,2}$ ” означало бы, что квалификационный матч по хоккею с шайбой между Францией и Польшей, проходивший на прошлой неделе, закончился со счетом 1:1. Безусловно, что при отсутствии в языке представления свойства композиционности функционирование системы формирования рассуждений значительно затрудняется.

Как было показано в главе 7, пропозициональная логика не обладает достаточной выразительной мощью, которая позволяла бы кратко описывать среду с много-

численными объектами. Например, мы были вынуждены записывать отдельное правило, которое связывает между собой наличие ветерка и ям для каждого квадрата, например, таким образом:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

С другой стороны, на естественном языке, по-видимому, совсем несложно раз и навсегда сформулировать правило: “В квадратах, непосредственно примыкающих к ямам, чувствуется ветерок”. Синтаксис и семантика естественного языка каким-то образом позволяют кратко описать эту среду.

Даже не очень глубокие размышления наводят на мысль, что естественные языки (такие как английский или русский) действительно являются очень выразительными. Например, авторы смогли написать почти всю эту книгу на естественном языке, лишь время от времени прибегая к использованию формальных языков (включая логику, математику и язык схем). В лингвистике и философии языка давно существует традиция, в которой естественный язык по сути рассматривается как декларативный язык представления знаний и предпринимаются попытки однозначно определить его формальную семантику. Если бы такая программа исследований завершилась успехом, это имело бы большое значение для искусственного интеллекта, поскольку позволило бы непосредственно использовать естественный язык (или какую-то его производную) в системах представления и формирования рассуждений.

В соответствии с современными взглядами естественный язык выполняет немного иное назначение: служит в качестве средства **общения**, а не исключительно как средство представления. Когда один приятель указывает пальцем и говорит другому: “Смотри!”, он хочет этим сказать, что, допустим, в фильме Супермен наконец-то взлетел над крышами. Тем не менее нельзя утверждать, что в предложении “Смотри!” закодирован именно этот факт. Вместо этого смысл предложения на естественном языке зависит и от самого предложения, и от **контекста**, в котором оно было высказано. Очевидно, что невозможно сохранить такое предложение, как “Смотри！”, в базе знаний и надеяться на то, что удастся восстановить его смысл, если в базе знаний не будет также представлен контекст. А из этого следует вопрос — как же можно представить сам контекст? К тому же естественные языки являются некомпозиционными — смысл предложения, такого как “Потом она это увидела”, может зависеть от контекста, состоящего из многих предшествующих и последующих предложений. Наконец, недостатком естественных языков является **неоднозначность**, которая может стать причиной затруднений при формировании рассуждений. Например, Пинкер [1212] выразил эту мысль таким образом: “Когда люди говорят о косе, то их, безусловно, не затрудняет определение того, идет ли речь об элементе женской прически, или о форме береговой полосы, или о чем-то другом, а поскольку одно и то же слово может соответствовать двум идеям, сами идеи не могут выражаться отдельными словами”.

### ЯЗЫК МЫШЛЕНИЯ

Философы и психологи уже в течение долгих столетий размышляют над тем, как люди и другие животные представляют знания. Очевидно, что в развитии этой способности у людей важную роль играла эволюция естественного языка. С другой стороны, многие психологические факты свидетельствуют о

том, что люди не используют язык непосредственно в своих внутренних представлениях. Например, кто сейчас может вспомнить, с какой именно из этих двух фраз начинался раздел 8.1?

- “В данном разделе рассматривается общий характер языков представления ...”
- “В данном разделе рассматривается тема, касающаяся языков представления знаний ...”

Ваннер [1553] обнаружил, что участники его экспериментов делали правильный выбор в подобных тестах на уровне случайности (выбирали правильный вариант примерно в 50% попыток), но помнили содержание того, что было ими прочитано с точностью больше чем 90%. Эти данные свидетельствуют о том, что люди обрабатывают слова, а затем формируют своего рода несловесное представление, которое мы называем **памятью**.

Вопрос о том, какими именно являются механизмы, с помощью которых люди используют язык для обеспечения и формирования представлений различных идей, продолжает привлекать значительный интерес. В знаменитой **гипотезе Сапира-Уорфа** утверждается, что язык, на котором мы говорим, оказывает глубокое влияние на тот способ, с помощью которого мы мыслим и принимаем решения, в частности, используемый язык влияет на формирование структуры категорий, с помощью которых мы подразделяем мир на объекты разных типов. Уорф [1585] утверждал, что эскимосы имеют в своем языке много слов для описания снега и поэтому снег вызывает у них более разнообразные восприятия по сравнению с людьми, говорящими на других языках. Некоторые лингвисты оспаривали фактографическую основу для этого утверждения (например, Паллем [1242] показал, что в таких эскимосских языках, как юпик и инупик, и в других родственных языках северных народов, по-видимому, имеется такое же количество слов для выражения понятий, касающихся снега, как и в английском языке), а другие выступили в его поддержку [485]. Однако, по-видимому, нельзя оспаривать ту мысль, что популяции, имеющие более глубокое знакомство с некоторыми аспектами мира, разрабатывают гораздо более подробные словари для описания этих аспектов. Например, энтомологи, проводящие полевые исследования, подразделяют то, что большинство из нас называет просто жуками, на сотни тысяч видов и лично знакомы с многими из них. (Биолог Дж.Б.С. Холдейн, изучающий эволюцию насекомых, когда-то пожаловался на то, что Создатель проявляет “невероятную заботу о жуках”.) Более того, множеством терминов для обозначения снега пользуются и опытные лыжники (различая такие состояния, как пудра, каша, пюре, простокваша, крупа, цемент, наст, сахар, асфальт, вельвет, пух, жижа и т.д.), которые в основном не известны обычному человеку. Однако остается неясным направление развития причинно-следственных отношений — узнают ли лыжники об этих различиях, только изучая свои профессиональные термины, или эти различия выявляются из индивидуального опыта и становятся согласованными с теми их обозначениями, которые в данное время приняты в этом сообществе спортсменов? Данный вопрос становится очень важным в исследованиях развития детей. До сих пор мы слишком мало знаем о том, насколько связаны друг с другом обучение языку и обучение мышлению. Например, позволяют ли знания о том,

как именуется некоторое понятие, такое как *бакалавр*, упростить формирование идей и размышление с помощью более сложных понятий, которые включают имя исходного понятия, например *претендент на звание бакалавра*?

Принятый авторами подход состоит в том, что в качестве основы следует выбрать пропозициональную логику, обладающую декларативной, композиционной семантикой, которая является независимой от контекста и непротиворечивой, и создать на этой основе более выразительную логику, заимствуя идеи представления из естественного языка и избегая вместе с тем его недостатков. Изучение синтаксиса естественного языка показывает, что наиболее очевидными его элементами являются существительные и именные конструкции, которые обозначают **объекты** (квадраты, ямы, вампузы), а также глаголы и глагольные конструкции, которые обозначают **отношения** между объектами (чувствовать ветерок, быть соседним, стрелять). Некоторые из этих отношений являются **функциями** — отношениями, в которых в ответ на каждое “входное” значение формируется только одно “выходное” значение. Такой подход позволяет сразу же приступить к составлению перечня примеров объектов, отношений и функций, как показано ниже.

- Объекты: люди, дома, числа, теории, Рональд Макдональд, цвета, бейсбольные соревнования, войны, столетия ...
- Отношения: могут быть унарными отношениями, или **свойствами**, такими как красный, круглый, поддельный, первичный, многоэтажный и т.д., либо более общими **n-арными** отношениями, такими как “быть братьями”, “быть больше”, “находиться внутри”, “входить в состав”, “иметь цвет”, “произойти позже”, “принадлежать”, “находиться между” и т.д.
- Функции: “быть отцом”, “быть лучшим другом”, “быть третьей подачей мяча”, “быть на единицу больше”, “быть началом” и т.д.

Безусловно, почти каждое утверждение может рассматриваться как обозначающее объекты, а также их свойства или отношения. Ниже приведены некоторые примеры.

- “Один плюс два равняется трем”.

Объекты: один, два, три, один плюс два; отношение: равняется; функция: плюс. (“Один плюс два” представляет собой название объекта, полученного путем применения функции “плюс” к объектам “один” и “два”. “Три” — другое название для этого объекта.)

- “В квадратах, соседних с тем квадратом, где находится вампус, чувствуется неприятный запах”.

Объекты: вампус, квадраты; свойство: неприятный запах; отношение: быть соседним.

- “Злой король Джон управлял Англией в 1200 году”.

Объекты: Джон, Англия, 1200 год; отношение: управлял; свойства: злой, король.

**Язык логики первого порядка**, синтаксис и семантику которого мы определим в следующем разделе, основан на понятиях объектов и отношений. Он стал чрезвычайно важным для математики, философии и искусственного интеллекта именно потому, что эти области знаний (а фактически основная часть повседневного человеческого существования) могут вполне продуктивно рассматриваться как касающиеся объектов

и отношений между ними. Логика первого порядка позволяет также выражать факты о некоторых или обо всех объектах во Вселенной. Это дает возможность представлять общие законы, или правила, такие как следующее утверждение: “В квадратах, соседних с тем квадратом, где находится вампус, чувствуется неприятный запах”.

Основное различие между пропозициональной логикой и логикой первого порядка заключается в том, что каждый из этих языков вносит различный **онтологический вклад** в описание действительности, т.е. они по-разному представляют характер действительности. Например, в пропозициональной логике предполагается, что существуют лишь факты, которые относятся или не относятся к данному миру. Каждый факт может находиться в одном из двух состояний: быть истинным или ложным<sup>2</sup>. В логике первого порядка приняты более широкие предположения, а именно, что мир состоит из объектов, между которыми могут быть или не быть некоторые отношения. Некоторые варианты логики специального назначения позволяют внести еще больший онтологический вклад; например, во **временной логике** предполагается, что факты имеют место в конкретные интервалы времени и что эти интервалы (которые могут рассматриваться как бесконечно малые или конечные) являются упорядоченными. Поэтому в вариантах логики специального назначения придается первоклассный статус некоторым видам объектов особого рода (и аксиомам об этих объектах), а не просто вводятся в базу знаний их определения. В **логике высокого порядка** как объекты трактуются сами отношения и функции, рассматриваемые в логике первого порядка. Это позволяет формировать утверждения обо всех отношениях, например, если потребуется точно определить, что означает понятие транзитивного отношения. В отличие от большинства вариантов логики специального назначения, логика высокого порядка является строго более выразительной, чем логика первого порядка, в том смысле, что некоторые высказывания логики высокого порядка не могут быть выражены с помощью любого конечного количества высказываний логики первого порядка.

Логику можно также охарактеризовать по ее **эпистемологическому вкладу** в познание; под этим подразумеваются возможные состояния знаний, которые она позволяет выразить в отношении каждого факта. И в пропозициональной логике, и в логике первого порядка любое высказывание представляет собой факт, и агент либо доверяет утверждению, что это высказывание истинно, либо доверяет утверждению, что оно ложно, либо не имеет мнения на этот счет. Поэтому в таких вариантах логики имеются три возможных состояния знаний, касающихся любого высказывания. С другой стороны, в тех системах, где используется **теория вероятностей**, может рассматриваться любая степень доверия<sup>3</sup>, начиная от 0 (полное недоверие) и заканчивая 1 (полное доверие). Например, в вероятностном мире вампуса агент может доверять утверждению о том, что вампус находится в квадрате  $[1, 3]$ , с вероятностью 0,75. Онтологический и эпистемологический вклады пяти различных вариантов логики показаны в табл. 8.1.

<sup>2</sup> В отличие от этого, в **нечеткой логике** факты имеют определенную **степень истинности** от 0 до 1. Например, в рассматриваемом мире высказывание “Вена — большой город” может считаться истинным только со степенью 0,6.

<sup>3</sup> Важно не путать степень доверия в теории вероятностей со степенью истинности в нечеткой логике. В действительности в некоторых системах нечеткой логики допускается выражать степень недоверия (или степень доверия) в отношении степеней истинности.

**Таблица 8.1. Формальные языки и показатели их онтологического и эпистемологического вкладов в познание**

Язык	Онтологический вклад (что существует в мире)	Эпистемологический вклад (какую степень доверия может выражать агент в отношении фактов)
Пропозициональная логика	Факты	Истинно/ложно/неизвестно
Логика первого порядка	Факты, объекты, отношения	Истинно/ложно/неизвестно
Временная логика	Факты, объекты, отношения, интервалы времени	Истинно/ложно/неизвестно
Теория вероятностей	Факты	Степень доверия $\in [0, 1]$
Нечеткая логика	Факты со степенью истинности $\in [0, 1]$	Известное интервальное значение

В следующем разделе мы приступим к изучению подробных сведений о логике первого порядка. Так же как студенту-физику требуется определенное знакомство с высшей математикой, так и студент, изучающий искусственный интеллект, должен развить у себя способность работать с логическими обозначениями. С другой стороны, важно также не слишком увлекаться изучением специфики отдельной логической системы обозначений, поскольку в конечном итоге количество различных версий таких систем исчисляется десятками. Главное, не упустить из виду то, благодаря чему данный язык обеспечивает формирование кратких представлений и таким образом его семантика приводит к созданию непротиворечивых процедур формирования рассуждений.

## 8.2. СИНТАКСИС И СЕМАНТИКА ЛОГИКИ ПЕРВОГО ПОРЯДКА

Начнем изложение темы этого раздела с более точного определения того способа, с помощью которого возможные миры логики первого порядка отражают ее онтологический вклад в познание объектов и отношений. Затем опишем различные элементы языка этой логики, в ходе этого объясняя их семантику.

### Модели для логики первого порядка

Как было указано в главе 7, моделями для любого логического языка служат формальные структуры, из которых состоят возможные рассматриваемые миры. Модели для пропозициональной логики представляют собой множества истинностных значений для пропозициональных символов. Модели для логики первого порядка являются более интересными. Прежде всего, в них имеются объекты!  **Проблемной областью** модели является множество объектов, которые она содержит; эти объекты иногда называют  **элементами проблемной области**. На рис. 8.1 показана модель с пятью объектами: Ричард Львиное Сердце, король Англии, который правил с 1189 по 1199 годы; его младший брат, злой король Джон, который правил с 1199 по 1215; левые ноги Ричарда и Джона; корона.

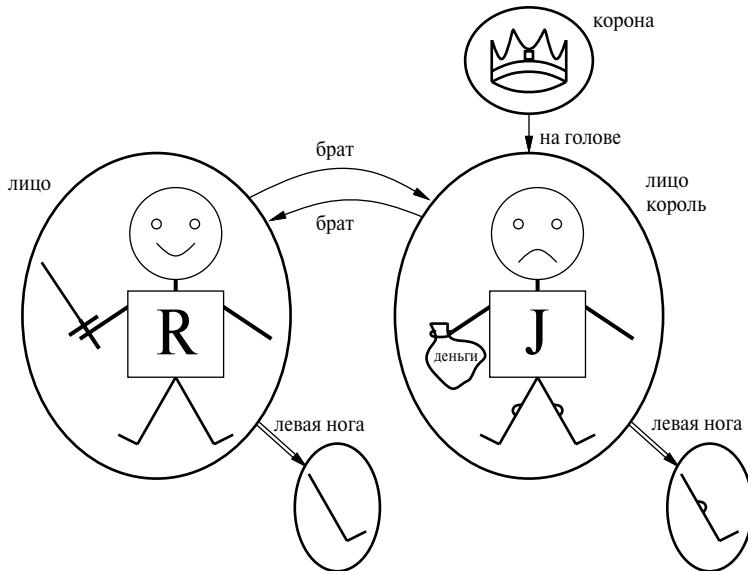


Рис. 8.1. Модель, состоящая из пяти объектов, двух бинарных отношений, трех унарных отношений (обозначенных метками на объектах) и одной унарной функции, “быть левой ногой”

Отношения между объектами в этой модели могут быть определены различными способами. На данном рисунке показано, что Ричард и Джон — братья. Выражаясь формально, отношение — это просто множество ~~кортежей~~ кортежей объектов, связанных друг с другом. (Кортеж — это коллекция объектов, расположенных в установленном порядке и записанных в угловых скобках, окружающих эти объекты.) Таким образом, отношение, обозначающее родство на уровне братьев в этой модели, представляет собой следующее множество:

$$\{<\text{Ричард Львинае Сердце}, \text{король Джон}>, \\ <\text{король Джон}, \text{Ричард Львинае Сердце}>\} \quad (8.1)$$

(Здесь эти объекты были указаны на естественном языке, но читатель при желании может мысленно вставить вместо имен этих королей их портреты.) Корона находится на голове короля Джона, поэтому отношение “быть на голове” содержит только один кортеж, *<корона, король Джон>*. Отношения “быть братом” и “быть на голове” являются бинарными, т.е. они устанавливают связь между парами объектов. Кроме того, эта модель содержит унарные отношения, или свойства: свойство “быть человеком” является истинным и для Ричарда, и для Джона; свойство “быть королем” истинно только для Джона (предположительно потому, что к этому моменту Ричард уже был мертв); а свойство “быть короной” истинно только для короны.

Некоторые виды связей удобнее рассматривать как функции, в том смысле, что указанным образом данный конкретный объект должен быть связан только с одним объектом. Например, каждый человек имеет только одну левую ногу, поэтому в данной модели имеется унарная функция “быть левой ногой”, которая включает следующие отображения:

$$<\text{Ричард Львинае Сердце}> \rightarrow \text{левая нога Ричарда} \\ <\text{король Джон}> \rightarrow \text{левая нога Джона} \quad (8.2)$$

Строго говоря, для модели в логике первого порядка требуются **полностью определенные функции**, т.е. функции, в которых должно быть предусмотрено значение для каждого входного кортежа. Таким образом, левую ногу должна иметь и корона, а также, безусловно, каждая из левых ног. Предусмотрено некоторое формальное решение этой неприятной проблемы, возникающей из-за того, что каждый объект, который в действительности не имеет левую ногу, включая саму левую ногу, в результате применения функции получает дополнительный “невидимый” объект, представляющий собой левую ногу. К счастью, при условии, что никто не будет составлять утверждений о левых ногах объектов, не имеющих левой ноги, применять эти формальные решения не обязательно.

## Символы и интерпретации

Вернемся к синтаксису языка. Нестерпеливый читатель может найти полное описание формальной грамматики логики первого порядка в листинге 8.1.

**Листинг 8.1. Синтаксис логики первого порядка с оператором равенства, заданный в форме Бэкуса-Наура.** (Для ознакомления с этой системой обозначений см. с. 1297.) В этом синтаксисе предусмотрены строгие правила применения круглых скобок; примечания, касающиеся круглых скобок и предшествования операторов, приведенные на с. 296, равным образом относятся и к логике первого порядка

---

```

Sentence → AtomicSentence
| ( Sentence Connective Sentence )
| Quantifier Variable, ... Sentence
| ~Sentence

AtomicSentence → Predicate(Term, ...) | Term = Term

Term → Function(Term, ...)
| Constant
| Variable

Connective → ⇒ | ∧ | ∨ | ⇔
Quantifier → ∀ | ∃
Constant → A | X1 | John | ...
Variable → a | x | s | ...
Predicate → Before | HasColor | Raining | ...
Function → Mother | LeftLeg | ...

```

---

Основными синтаксическими элементами логики первого порядка являются символы, которые обозначают объекты, отношения и функции. Поэтому сами символы подразделяются на три типа: **константные символы**, которые обозначают объекты; **предикатные символы**, которые обозначают отношения, и **функциональные символы**, которые обозначают функции. Примем соглашение, что имена этих символов будут начинаться с прописных букв. Например, могут использоваться константные символы *Richard* и *John*; предикатные символы *Brother*, *OnHead*, *Person*, *King* и *Crown* и функциональный символ *LeftLeg*. Как и применительно к пропозициональным символам, выбор имен этих символов полностью предоставляется пользователю.

телю. Каждый предикатный и функциональный символ характеризуется **арностью**, которая определяет количество формальных параметров.

Семантика должна связывать высказывания с моделями, для того чтобы можно было определить истинность. Чтобы иметь возможность решить такую задачу, требуется **интерпретация**, которая определяет, на какие именно объекты, отношения и функции ссылаются те или иные константные, предикатные и функциональные символы. Одна из возможных интерпретаций для рассматриваемого примера (которую мы будем называть **намеченной интерпретацией**) состоит в следующем:

- Символ *Richard* обозначает Ричарда Львиное Сердце, а символ *John* — злого короля Джона.
- Символ *Brother* обозначает отношение родства между братьями, т.е. множество кортежей объектов, приведенное в уравнении 8.1; символ *OnHead* обозначает отношение “быть на голове”, которое установлено между короной и королем Джоном; символы *Person*, *King* и *Crown* относятся к множествам объектов, представляющих собой людей, королей и короны.
- Символ *LeftLeg* относится к функции “быть левой ногой”, т.е. к отображению, приведенному в уравнении 8.2.

Может быть много других возможных интерпретаций, связывающих эти символы с данной конкретной моделью. Например, одна интерпретация отображает символ *Richard* на корону, а символ *John* — на левую ногу короля Джона. В этой модели имеется пять объектов, поэтому существует 25 возможных интерпретаций только для константных символов *Richard* и *John*. Обратите внимание на то, что не все объекты имеют имя, например, в данной намеченной интерпретации не предусмотрены имена для короны или для ног. Возможно также, чтобы один объект имел несколько имен; примером такой интерпретации была бы интерпретация, в которой и символ *Richard*, и символ *John* относились бы к короне. Если читатель находит, что такая возможность приводит к путанице, напомним, что в пропозициональной логике вполне допустимо иметь модель, в которой высказывания *Cloudy* (пасмурно) и *Sunny* (солнечно) одновременно являются истинными; задача исключения из рассмотрения моделей, несовместимых с нашими знаниями, возлагается на базу знаний.

Истинность любого высказывания определяется с помощью некоторой модели и некоторой интерпретации символов этого высказывания. Поэтому логическое следствие, допустимость и другие свойства высказываний определяются в терминах всех возможных моделей и всех возможных интерпретаций. Важно отметить, что количество элементов проблемной области в каждой модели может быть неограниченным, например, элементами проблемной области могут быть целые числа или действительные числа. Поэтому не ограничено количество возможных моделей, как и количество интерпретаций. Проверка логического следствия путем перебора всех возможных моделей, которая была осуществимой в пропозициональной логике, в логике первого порядка больше не может применяться. Даже если количество рассматриваемых объектов ограничено, количество их комбинаций может быть очень большим. Например, при использовании символов, рассматриваемых в данном примере, существует приблизительно  $10^{25}$  комбинаций для проблемной области с пятью объектами (см. упр. 8.5).

## Термы

❖ **Терм** — это логическое выражение, которое относится к некоторому объекту. Поэтому константные символы также представляют собой термы, но не всегда удобно иметь отдельный символ для каждого отдельного объекта. Например, в естественном языке можно воспользоваться обозначением “левая нога короля Джона”, а не присваивать ноге короля имя. Именно для этого и нужны функциональные символы: вместо использования константного символа мы можем написать *LeftLeg(John)*. В общем случае сложный терм формируется с помощью функционального символа, за которым следует заключенный в круглые скобки список формальных параметров данного функционального символа. Важно помнить, что любой сложный терм — это некоторое подразумеваемое имя, выраженное в сложной форме. Это — не “вызов процедуры”, которая “возвращает значение”. Не существует такой процедуры *LeftLeg*, которая принимает на входе какого-то человека и возвращает его левую ногу. Мы можем рассуждать о левых ногах (например, формулировать общее правило, что каждый человек имеет таковую, и на основании этого делать вывод, что таковую должен иметь Джон), даже не предусматривая определение символа *LeftLeg*. Это — такая логическая операция, которая не может быть выполнена с помощью процедур в языках программирования<sup>4</sup>.

Формальное определение семантики термов является несложным. Рассмотрим терм  $f(t_1, \dots, t_n)$ . Функциональный символ  $f$  относится к некоторой функции в модели (назовем ее  $F$ ); термы с обозначением формальных параметров относятся к объектам данной проблемной области (назовем их  $d_1, \dots, d_n$ ); а сам терм в целом относится к объекту, представляющему собой значение функции  $F$ , применяемой к объектам  $d_1, \dots, d_n$ . Например, предположим, что функциональный символ *LeftLeg* относится к функции, показанной в уравнении 8.2, а символ *John* относится к королю Джону; в таком случае *LeftLeg(John)* относится к левой ноге короля Джона. Таким образом, интерпретация определяет для каждого терма соответствующий референт (объект, к которому относится данный терм).

## Атомарные высказывания

Как было указано выше, термы позволяют ссылаться на объекты, предикатные символы — на отношения, а при их совместном использовании формируются **атомарные высказывания**, позволяющие констатировать факты. Атомарное высказывание состоит из предикатного символа, за которым следует заключенный в круглые скобки список термов:

*Brother(Richard, John)*

---

<sup>4</sup> **λ-выражения** представляют собой удобные обозначения, которые дают возможность формировать новые функциональные символы “динамически”. Например, функция, которая формирует квадрат своего формального параметра, может записываться как  $(\lambda x \ x \times x)$  и применяться к формальным параметрам точно так же, как и любой другой функциональный символ.  $\lambda$ -выражение можно также определить и использовать как предикатный символ (см. главу 22). Точно такую же роль играет оператор *lambda* на языке Lisp. Следует отметить, что использование  $\lambda$ -выражений в такой форме формально не увеличивает выразительную мощь логики первого порядка, поскольку любое высказывание, которое включает  $\lambda$ -выражение, может быть преобразовано путем “вставки” в него соответствующих формальных параметров для получения эквивалентного высказывания.

В соответствии с намеченной интерпретацией, приведенной выше, это атомарное высказывание констатирует тот факт, что Ричард Львиное Сердце — брат короля Джона<sup>5</sup>. Атомарные высказывания могут включать в качестве фактических параметров сложные термы. Поэтому в высказывании

*Married(Father(Richard), Mother(John))*

утверждается, что отец Ричарда Львиное Сердце был женат на матери короля Джона (опять-таки при использовании подходящей интерпретации).

Любое ~~атомарное высказывание является истинным в данной конкретной модели~~ при данной конкретной интерпретации, если отношение, на которое ссылается его предикатный символ, соблюдается среди объектов, на которые ссылаются его параметры.

## Сложные высказывания

Для формирования более сложных высказываний, как и в пропозициональном исчислении, могут использоваться **логические связки**. Семантика высказываний, сформированных с помощью логических связок, идентична семантике, которая рассматривается в пропозициональной логике. Ниже приведены четыре высказывания, которые являются истинными в модели, показанной на рис. 8.1, при использовании рассматриваемой намеченной интерпретации

$\neg \text{Brother}(\text{LeftLeg}(\text{Richard}), \text{John})$   
 $\text{Brother}(\text{Richard}, \text{John}) \wedge \text{Brother}(\text{John}, \text{Richard})$   
 $\text{King}(\text{Richard}) \vee \text{King}(\text{John})$   
 $\neg \text{King}(\text{Richard}) \Rightarrow \text{King}(\text{John})$

## Кванторы

После определения логики, которая допускает использование объектов, становится вполне естественным стремление к созданию средств, позволяющих выражать свойства целых коллекций объектов, а не перебирать эти объекты по именам. Это позволяют сделать ~~кванторы~~ кванторы. Логика первого порядка включает два стандартных квантора, называемых кванторами *всеобщности* и *существования*.

### Применение квантора всеобщности ( $\forall$ )

Напомним, с какими трудностями мы сталкивались в главе 7, пытаясь выразить общие правила в пропозициональной логике. С другой стороны, в логике первого порядка квинтэссенцией становятся такие правила, как: “В квадратах, соседних с тем квадратом, где находится вампир, чувствуется неприятный запах” и “Все короли являются людьми”. Первое из этих правил будет рассматриваться в разделе 8.3, а второе правило, “Все короли являются людьми”, записывается в логике первого порядка следующим образом:

$\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$

Квантор  $\forall$  обычно произносится как “для всех ...”. (Следует помнить, что перевернутая буква “A” обозначает “all” — все.) Таким образом, в этом высказывании утверждается следующее: “Для всех  $x$ , если  $x$  — король, то  $x$  — человек”. Символ  $x$

---

<sup>5</sup> Как правило, мы будем придерживаться такого соглашения об упорядочении параметров, что  $P(x, y)$  интерпретируется как “ $x$  представляет собой  $P$  от  $y$ ”.

называется **переменной**. В соответствии с общепринятым соглашением в качестве переменных применяются строчные буквы. Переменная сама является термом и как таковая может также служить параметром функции, например  $\text{LeftLeg}(x)$ . Терм без переменных называется **базовым термом**.

Интуитивно ясно, что в высказывании  $\forall x \ P$ , где  $P$  — любое логическое выражение, утверждается, что  $P$  является истинным для каждого объекта  $x$ . Точнее, высказывание  $\forall x \ P$  истинно в данной модели при данной интерпретации, если выражение  $P$  истинно при всех возможных **расширенных интерпретациях**, сформированных из данной интерпретации, где каждая расширенная интерпретация задает элемент проблемной области, на которую ссылается объект  $x$ .

На первый взгляд такое определение может показаться сложным, но оно фактически представляет собой лишь формальное определение интуитивного смысла применения квантора всеобщности. Рассмотрим модель, показанную на рис. 8.1, и намеченную интерпретацию, которая ее сопровождает. Эту интерпретацию можно расширить следующими пятью способами:

- $x \rightarrow$  Ричард Львиное Сердце
- $x \rightarrow$  король Джон
- $x \rightarrow$  левая нога Ричарда
- $x \rightarrow$  левая нога Джона
- $x \rightarrow$  корона

Высказывание с квантором всеобщности  $\forall x \ \text{King}(x) \Rightarrow \text{Person}(x)$  является истинным при первоначальной интерпретации, если высказывание  $\text{King}(x) \Rightarrow \text{Person}(x)$  истинно в каждой из пяти расширенных интерпретаций. Это означает, что данное высказывание с квантором всеобщности эквивалентно утверждению об истинности следующих пяти высказываний:

- Ричард Львиное Сердце — король  $\Rightarrow$  Ричард Львиное Сердце — человек
- король Джон — король  $\Rightarrow$  король Джон — человек
- левая нога Ричарда — король  $\Rightarrow$  левая нога Ричарда — человек
- левая нога Джона — король  $\Rightarrow$  левая нога Джона — человек
- корона — король  $\Rightarrow$  корона — человек

Рассмотрим внимательно это множество утверждений. Поскольку в нашей модели единственным королем является король Джон, то во втором высказывании утверждается, что он — человек, как и следовало ожидать. А что же можно сказать об остальных четырех высказываниях, в частности о тех, в которых приведены утверждения о ногах и коронах? Являются ли они частью смысла утверждения “Все короли являются людьми”? Действительно, остальные четыре утверждения истинны в данной модели, но не позволяют ничего судить о том, можно ли считать людьми ноги, короны или даже Ричарда. Это связано с тем, что ни один из этих объектов не является королем. Рассматривая истинностную таблицу для связки  $\Rightarrow$  (см. табл. 7.1), можно убедиться в том, что импликация истинна, даже если ее предпосылка ложна, независимо от того, является ли истинным заключение. Таким образом, утверждая истинность высказывания с квантором всеобщности, что эквивалентно утверждению об истинности целого списка отдельных импликаций, мы в конечном итоге утверждаем об истинности правила, выраженного в виде этого высказывания, только для тех объектов, для которых предпосылка является истинной, и вообще ничего не говорим о тех объектах, для которых предпосылка ложна. Поэтому, как оказалось,

записи истинностной таблицы, относящиеся к связке  $\Rightarrow$ , являются идеальным способом формулировки общих правил с кванторами всеобщности.

Распространенная ошибка, которую часто допускают даже внимательные читатели, которые прочли предыдущий абзац несколько раз, состоит в том, что они используют конъюнкцию вместо импликации. Тогда следующее высказывание:

$$\forall x \text{ King}(x) \wedge \text{Person}(x)$$

становится эквивалентным таким утверждениям:

Ричард Львиное Сердце – король  $\wedge$  Ричард Львиное Сердце – человек

король Джон – король  $\wedge$  король Джон – человек

левая нога Ричарда – король  $\wedge$  левая нога Ричарда – человек

и т.д. Очевидно, что такой ряд утверждений не передает желаемый смысл.

### Применение квантора существования ( $\exists$ )

Квантор всеобщности позволяет формировать утверждения о каждом объекте. Аналогичным образом, мы можем формировать утверждение о некотором объекте во вселенной без его именования с помощью квантора существования. Например, чтобы выразить мысль, что на голову короля Джона возложена корона, можно записать следующее:

$$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$$

Квантор существования с переменной  $\exists x$  читается как: “Существует  $x$ , такой, что ...”, или “Для некоторого  $x$  ...”.

Интуитивно ясно, что в высказывании  $\exists x P$  утверждается, будто выражение  $P$  истинно по меньшей мере для одного объекта  $x$ . Точнее, высказывание  $\exists x P$  истинно в данной конкретной модели при данной конкретной интерпретации, если выражение  $P$  истинно по меньшей мере в одной расширенной интерпретации, в которой присваивается  $x$  одному из элементов проблемной области. В данном примере это означает, что должно быть истинным по меньшей мере одно из приведенных ниже утверждений.

Ричард Львиное Сердце – корона  $\wedge$  Ричард Львиное Сердце находится  
на голове Джона

король Джон – корона  $\wedge$  король Джон находится на голове Джона

левая нога Ричарда – корона  $\wedge$  левая нога Ричарда находится  
на голове Джона

левая нога Джона – корона  $\wedge$  левая нога Джона находится на голове Джона  
корона – корона  $\wedge$  корона находится на голове Джона

В рассматриваемой модели истинно пятое утверждение, поэтому в ней является истинным само первоначальное утверждение с квантором существования. Обратите внимание на то, что в соответствии с приведенным выше определением квантора существования это высказывание будет также истинным и в такой модели, в которой на короля Джона возложены две короны. Такая ситуация является полностью совместимой с первоначальным высказыванием<sup>6</sup>: “На голову короля Джона возложена корона”.

<sup>6</sup> Применяется также определенный вариант квантора существования, обычно записываемый как  $\exists^1$  или  $\exists!$ , который означает: “Существует только один...”. Как будет показано в разделе 8.2, тот же смысл можно выразить с использованием утверждений, содержащих знак равенства.

Итак, логическая связка  $\Rightarrow$  может рассматриваться как наиболее подходящая для использования с квантором  $\forall$ , а логическая связка  $\wedge$  естественным образом подходит для использования с квантором  $\exists$ . В примере, который рассматривался в предыдущем разделе, применение  $\wedge$  в качестве основной связки в сочетании с квантором  $\forall$  приводило к формированию слишком сильного утверждения, а использование связки  $\Rightarrow$  в сочетании с квантором  $\exists$  обычно приводит к формированию действительно очень слабых утверждений. Рассмотрим следующее высказывание:

$$\exists x \text{ Crown}(x) \Rightarrow \text{OnHead}(x, John)$$

На первый взгляд может показаться, что в этом высказывании вполне успешно передана мысль о том, что на голову короля Джона возложена корона. Применяя соответствующее определение семантики, можно убедиться в том, что данное высказывание декларирует истинность по меньшей мере одного из следующих утверждений:

$$\begin{aligned} \text{Ричард Львиное Сердце} - \text{корона} &\Rightarrow \text{Ричард Львиное Сердце находится} \\ &\quad \text{на голове Джона} \end{aligned}$$

$$\text{король Джон} - \text{корона} \Rightarrow \text{король Джон находится на голове Джона}$$

$$\begin{aligned} \text{левая нога Ричарда} - \text{корона} &\Rightarrow \text{левая нога Ричарда находится} \\ &\quad \text{на голове Джона} \end{aligned}$$

и т.д. Итак, импликация истинна, если и предпосылка, и заключение являются истинными, или если ложна ее предпосылка. Поэтому, если Ричард Львиное Сердце — не корона, то первое утверждение истинно и выполняется высказывание с квантором существования. Таким образом, высказывание в форме импликации с квантором существования истинно в любой модели, содержащей объект, для которого предпосылка импликации является ложной, поэтому подобные высказывания фактически не несут почти никакой информации.

### Вложенные кванторы

Часто возникает необходимость сформировать более сложные высказывания с использованием нескольких кванторов. Простейшим является случай, когда кванторы относятся к одному и тому же типу. Например, утверждение: “Братья — это люди, связанные братскими родственными узами”, может быть записано следующим образом:

$$\forall x \forall y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$$

Последовательно применяемые кванторы могут быть записаны как один квантор с несколькими переменными. Например, чтобы выразить мысль о том, что родственные отношения между людьми, связанными братскими узами, являются симметричным, можно составить следующее высказывание:

$$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$$

В других случаях возникает необходимость в использовании сочетания разных кванторов. Например, строка из песни “Everybody loves somebody” (Каждый кого-то любит) означает, что для каждого человека существует кто-то, кого этот человек любит:

$$\forall x \exists y \text{ Loves}(x, y)$$

С другой стороны, чтобы сформулировать утверждение “Есть некто, кого любят все”, можно записать следующее:

$$\forall y \exists x \text{ Loves}(x, y)$$

Таким образом, порядок расположения кванторов очень важен. Он становится очевиднее после вставки круглых скобок. В высказывании  $\forall x (\exists y Loves(x, y))$  утверждается, что каждый имеет конкретное свойство, а именно то свойство, что его кто-то любит. С другой стороны, в высказывании  $\exists x (\forall y Loves(x, y))$  утверждается, что некто в мире имеет конкретное свойство, а именно свойство быть любимым всеми.

Если два квантора используются с одним и тем же именем переменной, может возникнуть некоторая путаница. Рассмотрим следующее высказывание:

$$\forall x [Crown(x) \vee (\exists x Brother(Richard, x))]$$

Здесь к переменной  $x$  в атомарном высказывании  $Brother(Richard, x)$  применяется квантор существования. Общее правило состоит в том, что переменная принадлежит к самому внутреннему квантору, в котором она упоминается; это означает, что такая переменная не может стать субъектом действия любого другого квантора<sup>7</sup>. Еще один способ анализа приведенного выше высказывания состоит в следующем:  $\exists x Brother(Richard, x)$  — это высказывание о Ричарде (о том, что у него есть брат), а не о переменной  $x$ , поэтому размещение квантора  $\forall x$  за пределами данного высказывания не оказывает на него никакого действия, и оно могло быть равным образом записано как  $\exists z Brother(Richard, z)$ . Но поскольку такая ситуация может стать источником путаницы, в подобных обстоятельствах мы всегда будем использовать разные переменные.

### Связь между кванторами $\forall$ и $\exists$

Кванторы  $\forall$  и  $\exists$  фактически тесно связаны друг с другом через отрицание. Утверждение о том, что никто не любит пастернак, равносильно утверждению о том, что не существует никого, кто бы его любил, и наоборот:

$$\forall x \neg Likes(x, Parsnips) \text{ эквивалентно высказыванию } \neg \exists x Likes(x, Parsnips)$$

По такому же принципу может быть сформирована более сложная конструкция; например, выражение “Все любят мороженое” означает, что нет никого, кто не любил бы мороженое:

$$\forall x Likes(x, IceCream) \text{ эквивалентно высказыванию } \neg \exists x \neg Likes(x, IceCream)$$

Поскольку квантор  $\forall$  фактически определяет в универсуме объектов конъюнкцию, а квантор  $\exists$  определяет дизъюнкцию, нет ничего удивительного в том, что они подчиняются правилам де Моргана. Правила де Моргана для высказываний с кванторами и без кванторов приведены ниже.

$$\begin{aligned} \forall x \neg P &\equiv \neg \exists x P \\ \neg P \wedge \neg Q &\equiv \neg(P \vee Q) \end{aligned}$$

<sup>7</sup> Тем не менее сохраняется возможность обеспечить взаимодействие кванторов с использованием переменной с одним и тем же именем, которая послужила побудительной причиной разработки несколько замысловатого механизма применения расширенных интерпретаций в семантике высказываний с кванторами. В данном примере более интуитивно очевидный подход, при котором выполняется подстановка объектов вместо каждого вхождения переменной  $x$ , становится неосуществимым, поскольку значение переменной  $x$  в атомарном высказывании  $Brother(Richard, x)$  в результате такой подстановки будет “закреплено”. Расширенные интерпретации позволяют справиться с этой ситуацией правильно, поскольку присваивание значения переменной  $x$  во внутреннем кванторе перекрывает присваивание, выполняемое во внешнем кванторе.

$$\begin{aligned}\neg \forall x P &\equiv \exists x \neg P \\ \neg(P \wedge Q) &\equiv \neg P \vee \neg Q \\ \forall x P &\equiv \neg \exists x \neg P \\ P \wedge Q &\equiv \neg(\neg P \vee \neg Q) \\ \exists x P &\equiv \neg \forall x \neg P \\ P \vee Q &\equiv \neg(\neg P \wedge \neg Q)\end{aligned}$$

Таким образом, в действительности нет необходимости иметь одновременно кванторы  $\forall$  и  $\exists$ , так же как фактически не нужны обе связи  $\wedge$  и  $\vee$ . Тем не менее удобство для чтения важнее, чем экономия выразительных средств, поэтому мы будем пользоваться обоими этими кванторами.

## Равенство

В логике первого порядка предусмотрен еще один способ составления атомарных высказываний, отличный от использования предикатных символов и термов, как было описано выше. Для составления утверждений о том, что два терма ссылаются на один и тот же объект, может использоваться **символ равенства**. Например, следующее утверждение:

$$\text{Father}(\text{John}) = \text{Henry}$$

говорит о том, что объект, на который ссылается высказывание  $\text{Father}(\text{John})$ , и объект, указанный под именем  $\text{Henry}$ , представляет собой одно и то же. Поскольку в любой интерпретации за каждым термом закрепляется референт, т.е. объект, на который он ссылается, определение истинности любого высказывания с символом равенства сводится к проверке того, представляют ли собой референты двух термов, соединенных символом равенства, один и тот же объект.

Символ равенства может использоваться для констатации фактов, касающихся данной конкретной функции, как было только что сделано применительно к функциональному символу  $\text{Father}$ . Он может также применяться с отрицанием как указание на то, что два терма не представляют собой один и тот же объект. Чтобы выразить мысль о том, что Ричард имеет по меньшей мере двух братьев, можно записать следующее:

$$\exists x, y \text{Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard}) \wedge \neg(x = y)$$

С другой стороны, высказывание:

$$\exists x, y \text{Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard})$$

не имеет такого намеченного смысла. В частности, оно истинно в модели, приведенной на рис. 8.1, где у Ричарда имеется только один брат. Чтобы убедиться в этом, рассмотрим расширенную интерпретацию, в которой обеим переменным,  $x$  и  $y$ , присваивается объект — король Джон. Но в результате добавления выражения  $\neg(x=y)$  такие модели становятся недействительными. В качестве сокращения для  $\neg(x=y)$  иногда используется обозначение  $x \neq y$ .

## 8.3. ИСПОЛЬЗОВАНИЕ ЛОГИКИ ПЕРВОГО ПОРЯДКА

Теперь, после того как мы определили выразительный логический язык, можно приступить к изучению способов его использования. Лучше всего можно сделать это с помощью примеров. Выше уже были приведены некоторые простые высказыва-

ния, иллюстрирующие различные аспекты этого логического синтаксиса; в данном разделе будут показаны более систематические представления некоторых простых **проблемных областей**. В проблематике представления знаний проблемной областью считается некоторая часть мира, о которой необходимо выразить некоторые знания.

Начнем с краткого описания интерфейса Tell/Ask для базы знаний в логике первого порядка. Затем рассмотрим проблемную область семейных отношений, чисел, множеств и списков, а также мира вампуса. В следующем разделе приведен более развернутый пример (электронные схемы), а в главе 10 рассматривается все, что касается данной области науки.

### Утверждения и запросы в логике первого порядка

Высказывания вводятся в базу знаний с помощью операции Tell, точно так же, как и в пропозициональной логике. Такие высказывания называются **утверждениями**. Например, можно ввести утверждения, что Джон — король и что короли — люди:

```
Tell(KB, King(John))
Tell(KB,  $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$ )
```

Мы можем задавать вопросы о содержимом базы знаний с использованием операции Ask. Например, следующее выражение:

```
Ask(KB, King(John))
```

возвращает *true*. Вопросы, заданные с помощью операции Ask, называются **запросами**, или **целями** (которые не следует путать с целями, используемыми при описании желаемых состояний агента). Вообще говоря, на любой запрос, который логически следует из базы знаний, должен быть получен утвердительный ответ. Например, если в ней содержатся два утверждения, приведенные в предыдущем абзаце, то следующий запрос:

```
Ask(KB, Person(John))
```

должен также возвратить *true*. Кроме того, можно задавать запросы с кванторами, такие как следующий:

```
Ask(KB,  $\exists x \text{ Person}(x)$ )
```

На этот запрос должен быть получен ответ *true*, но этот ответ — ни полезный, ни забавный. (Его можно сравнить с получением ответа “Да” на вопрос: “Можете ли вы сказать мне, который час?”) Запрос с переменными, на которые распространяется квантор существования, имеет смысл: “Существует ли такое значение *x*, что...”, и мы решаем его, предоставляем соответствующее значение *x*. Стандартная форма для ответа такого рода представляет собой **подстановку**, или **список связывания**, который является множеством пар “переменная–терм”. В данном конкретном случае, при наличии только двух утверждений, ответом должно быть  $\{x/John\}$ . А если имеется больше одного возможного ответа, может быть возвращен список подстановок.

### Проблемная область родства

В качестве первого примера рассмотрим проблемную область семейных отношений, или родства. Эта проблемная область включает такие факты, как “Элизабет —

мать Чарльза” и “Чарльз — отец Уильяма”, и правила наподобие того, что “Бабушка — это мать родителя”.

Очевидно, что объектами в этой проблемной области являются люди. В ней будут применяться два унарных предиката, *Male* (Мужчина) и *Female* (Женщина). Отношения родства (связи между родителями и детьми, братьями и сестрами, мужем и женой и т.д.) будут представлены с помощью бинарных предикатов: *Parent* (Родитель), *Sibling* (Брат или сестра), *Brother* (Брат), *Sister* (Сестра), *Child* (Ребенок), *Daughter* (Дочь), *Son* (Сын), *Spouse* (Супруг или супруга), *Wife* (Жена), *Husband* (Муж), *Grandparent* (Дедушка или бабушка), *Grandchild* (Внук или внучка), *Cousin* (Двоюродный брат или двоюродная сестра), *Aunt* (Тетя) и *Uncle* (Дядя). В качестве предикатов *Mother* (Мать) и *Father* (Отец) мы будем использовать функции, поскольку каждый человек имеет по одному из этих объектов (по крайней мере, в соответствии с законами природы).

Рассмотрим каждую из функций и предикатов, записывая все, что мы знаем о них, в терминах других символов. Например, мать — это родитель женского рода:

$$\forall m, c \text{ } Mother(c) = m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c)$$

Муж — это супруг мужского пола:

$$\forall w, h \text{ } Husband(h, w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h, w)$$

Мужчины и женщины — непересекающиеся категории людей:

$$\forall x \text{ } \text{Male}(x) \Leftrightarrow \neg \text{Female}(x)$$

Отношения между родителями и детьми являются взаимно противоположными:

$$\forall p, c \text{ } \text{Parent}(p, c) \Leftrightarrow \text{Child}(c, p)$$

Дедушка или бабушка — это родитель родителя:

$$\forall g, c \text{ } \text{Grandparent}(g, c) \Leftrightarrow \forall p \text{ } \text{Parent}(g, p) \wedge \text{Parent}(p, c)$$

Брат или сестра — это еще один ребенок тех же родителей:

$$\forall x, y \text{ } \text{Sibling}(x, y) \Leftrightarrow x \neq y \wedge \forall p \text{ } \text{Parent}(p, x) \wedge \text{Parent}(p, y)$$

Формулируя подобные сведения, можно заполнить еще несколько страниц, и в упр. 8.11 предлагается сделать именно это.

Каждое из этих высказываний может рассматриваться как одна из **аксиом** в проблемной области родства. Аксиомы обычно принято связывать чисто с математическими проблемными областями (и мы вскоре рассмотрим некоторые аксиомы для чисел), но они нужны во всех проблемных областях. Аксиомы предоставляют основную фактическую информацию, на основании которой могут быть получены логическим путем полезные заключения. Кроме того, приведенные выше аксиомы родства имеют **определения**; последние представлены в форме  $\forall x, y \text{ } P(x, y) \Leftrightarrow \dots$ . Аксиомы определяют функцию *Mother* и предикаты *Husband*, *Male*, *Parent*, *Grandparent* и *Sibling* в терминах других предикатов. Но приведенные выше определения можно “свести” к базовому множеству предикатов (*Child*, *Spouse* и *Female*), в терминах которых в конечном итоге определяются все остальные предикаты. Это — очень естественный способ, с помощью которого создается представление некоторой проблемной области, и он аналогичен способу, применяемому при создании пакетов программного обеспечения путем последовательного определения процедур из при-

митивных библиотечных функций. Обратите внимание на то, что множество примитивных базовых предикатов не обязательно должно быть уникальным; с таким же успехом можно использовать множество *Parent*, *Spouse* и *Male*. Как будет показано ниже, в некоторых проблемных областях нельзя найти четко определимое базовое множество.

Не все логические высказывания о некоторой проблемной области являются аксиомами. Некоторые из них представляют собой **теоремы**, т.е. следуют из аксиом. Например, рассмотрим утверждение, что родственные отношения между братьями и сестрами являются симметричными:

$$\forall x, y \ Sibling(x, y) \Leftrightarrow Sibling(y, x)$$

Это — аксиома или теорема? В действительности это — теорема, которая логически следует из аксиомы, определяющей понятие родственных отношений между братьями и сестрами. Если в базу знаний с помощью операции *Ask* будет введен запрос в виде этого высказывания, то должно быть получено значение *true*.

С чисто логической точки зрения в базе знаний должны содержаться только аксиомы, но не теоремы, поскольку теоремы не увеличивают множество заключений, которые следуют из базы знаний. Но с практической точки зрения важным свойством теорем является то, что они уменьшают вычислительные издержки на логический вывод новых высказываний. Без них системе формирования рассуждений приходится всякий раз начинать с самых фундаментальных принципов, во многом аналогично тому, что математику приходилось бы снова выводить правила исчисления, приступая к решению каждой новой задачи.

Не все аксиомы имеют определения. Некоторые из них предоставляют более общую информацию об определенных предикатах без формулировки определений. И действительно, некоторые предикаты не имеют полного определения, поскольку мы не знаем достаточно для того, чтобы их полностью охарактеризовать. Например, нельзя найти очевидного способа закончить следующее высказывание с определением понятия *человек*:

$$\forall x \ Person(x) \Leftrightarrow \dots$$

К счастью, логика первого порядка позволяет использовать предикат *Person*, не определяя его полностью. Вместо этого можно записывать частичные спецификации свойств, которыми обладает каждый человек, и свойства, которые идентифицируют некоторый объект как человека:

$$\forall x \ Person(x) \Rightarrow \dots$$

$$\forall x \dots \Rightarrow Person(x)$$

Аксиомы могут также представлять собой просто “голые факты”, такие как *Male(Jim)* и *Spouse(Jim, Laura)*. Такие факты формируют описания конкретных экземпляров задачи, что дает возможность находить ответы на конкретные вопросы. Ответы на эти вопросы представляют собой теоремы, которые следуют из аксиом. Но часто приходится убеждаться в том, что ожидаемые ответы получить не удается, например, можно было бы ожидать, что удастся вывести логическим путем факт *Female(Laura)* из аксиом *Male(George)* и *Spouse(George, Laura)*, но данный факт не следует как теорема из этих аксиом. Такая ситуация свидетельствует о том, что в базе знаний не хватает какой-то аксиомы. В упр. 8.8 предлагается предоставить эту аксиому.

## Числа, множества и списки

По-видимому, числа представляют собой наиболее яркий пример того, как может быть построена крупная теория из крошечного ядра аксиом. В этом разделе будет описана теория **натуральных чисел**, или неотрицательных целых чисел. Для этого потребуется предикат *NatNum*, который будет принимать истинное значение при использовании параметра, представляющего собой натуральное число; кроме того, требуется один константный символ, 0, и один функциональный символ, *S* (successor — преемник). **Аксиомы Пеано** определяют натуральные числа и операцию сложения<sup>8</sup>. Натуральные числа определяются рекурсивно:

$$\begin{aligned} \textit{NatNum}(0) \\ \forall n \textit{ NatNum}(n) \Rightarrow \textit{NatNum}(S(n)) \end{aligned}$$

Это означает, что 0 — натуральное число и для каждого объекта *n*, если *n* — натуральное число, *S(n)* — натуральное число. Поэтому натуральными числами являются 0, *S(0)*, *S(S(0))* и т.д. Необходимы также аксиомы для ограничения действия функции определения преемника:

$$\begin{aligned} \forall n \ 0 \neq S(n) \\ \forall m, n \ m \neq n \Rightarrow S(m) \neq S(n) \end{aligned}$$

Теперь мы можем определить операцию сложения в терминах функции определения преемника:

$$\begin{aligned} \forall m \textit{ NatNum}(m) \Rightarrow +(0, m) = m \\ \forall m, n \textit{ NatNum}(m) \wedge \textit{ NatNum}(n) \Rightarrow +(S(m), n) = S(+m, n) \end{aligned}$$

В первой из этих аксиом утверждается, что сложение числа 0 с любым натуральным числом *m* приводит к получению самого числа *m*. Обратите внимание на использование бинарного функционального символа “+” в терме  $+(0, m)$ ; в обычной математике такой терм принято записывать в виде  $0+m$  с использованием **инфиксной** системы обозначений. (Система обозначений, которая используется в этом разделе для логики первого порядка, называется **префиксной**.) Чтобы было удобнее читать высказывания, касающиеся чисел, здесь будет также разрешено использовать инфиксные обозначения. Кроме того, мы можем записывать как *S(n)* как *n+1*, поэтому вторая аксиома принимает следующий вид:

$$\forall m, n \textit{ NatNum}(m) \wedge \textit{ NatNum}(n) \Rightarrow (m+1)+n = (m+n)+1$$

Эта аксиома сводит сложение к повторному применению функции определения преемника.

Такое использование инфиксных обозначений представляет собой пример применения **синтаксического упрощения**, т.е расширения или сокращения стандартного синтаксиса, при котором семантика не изменяется. Любое высказывание, в котором используются такие сокращения, может быть “разупрощено” для получения эквивалентного высказывания в обычной логике первого порядка.

После определения понятия сложения становится простой задача определения умножения как повторного сложения, возведения в степень как повторного умно-

---

<sup>8</sup> Аксиомы Пеано включают также принцип индукции, который представляет собой высказывание логики второго порядка, а не логики первого порядка. Важность этого различия объясняется в главе 9.

жения, целочисленного деления и определения остатков от деления, формулировки понятия простых чисел и т.д. Таким образом, вся теория чисел (включая такие ее приложения, как криптография) может быть сформирована на основе одной константы, одной функции, одного предиката и четырех аксиом.

Проблемная область **множеств** также является фундаментальной не только для математики, но и для рассуждений на уровне здравого смысла. (В действительности на основе теории множеств может быть также построена теория чисел.) Необходимо иметь возможность представлять отдельные множества, включая пустое множество. Кроме того, требуется способ составления множеств с помощью добавления элемента к множеству или применения операции объединения или пересечения к двум множествам. К тому же необходимо знать, входит ли некоторый элемент в состав множества, и иметь возможность отличать множества от объектов, не являющихся множествами.

В качестве синтаксического упрощения в данном разделе будет использоваться обычный словарь теории множеств. Пустое множество представляет собой константу, которая записывается как  $\{\}$ . Применяется также один унарный предикат, *Set*, который принимает истинное значение, если его фактическим параметром является множество. Бинарными предикатами являются  $x \in s$  ( $x$  — элемент множества  $s$ ) и  $s_1 \subseteq s_2$  ( $s_1$  — подмножество, не обязательно строгое, множества  $s_2$ ). В качестве бинарных функций применяются  $s_1 \cap s_2$  (пересечение двух множеств),  $s_1 \cup s_2$  (объединение двух множеств) и  $\{x | s\}$  (множество, полученное в результате присоединения элемента  $x$  к множеству  $s$ ). Один из возможных наборов аксиом приведен ниже.

1. Единственными множествами являются пустое множество и множества, полученные путем присоединения некоторого элемента к множеству.

$$\forall s \ Set(s) \Leftrightarrow (s = \{\}) \vee (\forall x, s_2 \ Set(s_2) \wedge s = \{x | s_2\})$$

2. Пустое множество не имеет присоединенных к нему элементов, иными словами, не существует способа разложения множества *EmptySet* на меньшее множество и еще один элемент:

$$\neg \forall x, s \ \{x | s\} = \{\}$$

3. Присоединение к множеству элемента, уже имеющегося в этом множестве, не оказывает никакого эффекта:

$$\forall x, s \ x \in s \Leftrightarrow s = \{x | s\}$$

4. Единственными элементами множества являются элементы, которые были к нему присоединены. Мы выражим эту мысль рекурсивно, утверждая, что  $x$  — элемент множества  $s$  тогда и только тогда, когда  $s$  эквивалентно некоторому множеству  $s_2$ , к которому присоединен некоторый элемент  $y$ , где либо  $y$  совпадает с  $x$ , либо  $x$  является элементом  $s_2$ :

$$\forall x, s \ x \in s \Leftrightarrow [\forall y, s_2 \ (s = \{y | s_2\} \wedge (x = y \vee x \in s_2))]$$

5. Множество является подмножеством другого множества тогда и только тогда, когда все элементы первого множества являются элементами второго множества:

$$\forall s_1, s_2 \ s_1 \subseteq s_2 \Leftrightarrow (\forall x \ x \in s_1 \Rightarrow x \in s_2)$$

6. Два множества равны тогда и только тогда, когда каждое из них является подмножеством другого:

$$\forall s_1, s_2 \ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

7. Некоторый объект принадлежит к пересечению двух множеств тогда и только тогда, когда он является элементом обоих этих множеств:  
 $\forall x, s_1, s_2 \ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$
8. Некоторый объект принадлежит к объединению двух множеств тогда и только тогда, когда он является элементом того или другого множества:  
 $\forall x, s_1, s_2 \ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

☞ **Списки** подобны множествам. Различия между ними состоят в том, что списки упорядочены, а один и тот же элемент может появляться в списке несколько раз. Для списков может использоваться словарь ключевых слов Lisp: *Nil* — это список-константа без элементов; *Cons*, *Append*, *First* и *Rest* — функции; *Find* — предикат, который выполняет для списков такие же функции, какие *Member* выполняет для множеств. *List?* — предикат, принимающий истинное значение только при получении параметра, представляющего собой список. Как и в случае множеств, обычно принято использовать синтаксические упрощения в логических высказываниях, в которых применяются списки. Пустой список обозначается как `[ ]`. Терм *Cons*(*x*, *y*), где *y* — непустой список, записывается как `[x | y]`. Терм *Cons*(*x*, *Nil*) (т.е. список, содержащий только элемент *x*) записывается как `[x]`. Список из нескольких элементов, такой как `[A, B, C]`, соответствует вложенному терму *Cons*(*A*, *Cons*(*B*, *Cons*(*C*, *Nil*))) . В упр. 8.14 предлагается составить аксиомы для списков.

## Мир вампуса

Некоторые аксиомы пропозициональной логики для мира вампуса были приведены в главе 7. Аксиомы первого порядка, рассматриваемые в этом разделе, являются гораздо более краткими и выражают совершенно естественным способом именно то, что требуется описать в этом мире.

Напомним, что агент в мире вампуса получает вектор восприятий с пятью элементами. Соответствующее высказывание в логике первого порядка, хранящееся в базе знаний, должно включать данные о результатах восприятия и о времени, в которое они были получены; в противном случае у агента возникнет путаница в отношении того, когда и что он воспринимал. Для обозначения интервалов времени будут использоваться целые числа. Типичным высказыванием с данными о восприятии является следующее:

*Percept*(`[Stench, Breeze, Glitter, None, None]`, 5)

где *Percept* — бинарный предикат; *Stench* и т.д. — константы, помещенные в список. Действия в мире вампуса могут быть представлены с помощью логических термов следующим образом:

*Turn*(*Right*), *Turn*(*Left*), *Forward*, *Shoot*, *Grab*, *Release*, *Climb*

Чтобы определить, какое действие является наилучшим, программа агента составляет примерно такой запрос:

За *BestAction*(*a*, 5)

Функция *Ask* должна разрешить этот запрос и возвратить список связывания, такой как `{a / Grab}`. Затем программа агента может вернуть *Grab* как действие, которое

должно быть выполнено, но вначале должна ввести в свою собственную базу знаний данные о том, что будет выполнено действие *Grab*, с помощью операции *Tell*.

Из исходных данных о восприятии следуют некоторые факты о текущем состоянии, например:

$$\begin{aligned} \forall t, s, g, m, c \ Percept([s, Breeze, g, m, c], t) &\Rightarrow Breeze(t) \\ \forall t, s, b, m, c \ Percept([s, b, Glitter, m, c], t) &\Rightarrow Glitter(t) \end{aligned}$$

и т.д. Эти правила являются проявлением простейшей формы процесса формирования рассуждений, называемого **восприятием**, который будет подробно рассматриваться в главе 24. Обратите внимание на то, что квантификация происходит по переменной *t* с обозначением времени. А в пропозициональной логике приходилось создавать копии каждого высказывания для каждого интервала времени.

Кроме того, в этой логике могут быть реализованы простые “рефлекторные” варианты поведения с помощью импликационных высказываний с кванторами. Например, может быть предусмотрено следующее правило:

$$\forall t \ Glitter(t) \Rightarrow BestAction(Grab, t)$$

При наличии результатов восприятия и правил, приведенных в предыдущих абзацах, применение данного правила привело бы к желаемому заключению *BestAction(Grab, 5)* о том, что в данный момент следует выполнить действие *Grab*. Обратите внимание на соответствие между этим правилом и прямым соединением “восприятие/действие” в агенте на основе логической схемы, приведенной на рис. 7.9; соединение в этой логической схеме неявно предусматривает применение квантора к переменной, обозначающей время.

До сих пор в этом разделе все высказывания, касающиеся времени, были ~~с~~ **синхронными** (т.е. “одновременными”) высказываниями; это означает, что они связывали свойства некоторого состояния мира с другими свойствами того же состояния мира. А высказывания, которые допускают формирование “разновременных” рассуждений, называются ~~с~~ **диахронными**; например, агенту требуется знать, как комбинировать информацию о его предыдущем местонахождении с информацией о только что выполненном действии, чтобы определить свое текущее местонахождение. Отложим обсуждение диахронных высказываний до главы 10, а пока будем просто предполагать, что в отношении предикатов, касающихся изменения местонахождения, и других предикатов, зависящих от времени, выполняется требуемый логический вывод.

Выше были представлены восприятия и действия; теперь настало время представить саму среду. Начнем с объектов. Очевидными кандидатами являются квадраты, ямы и вампус. Можно было бы присвоить имя каждому квадрату (*Square<sub>1,2</sub>* и т.д.), но тогда тот факт, что *Square<sub>1,2</sub>* и *Square<sub>1,3</sub>* являются соседними, пришлось бы оформить как “дополнительный” факт и нам потребовалось по одному такому факту для каждой пары квадратов. Поэтому лучше использовать сложный терм, в котором строка и столбец показаны в виде целых чисел; например, списоковый терм [1, 2]. В таком случае определение понятия соседства любых двух квадратов можно представить следующим образом:

$$\begin{aligned} \forall x, y, a, b \ Adjacent([x, y], [a, b]) &\Leftrightarrow \\ [a, b] &\in \{[x+1, y], [x-1, y], [x, y+1], [x, y-1]\} \end{aligned}$$

Кроме того, можно было бы присвоить имя каждой яме, но такое решение является неподходящим по другой причине: нам нет смысла проводить различия между

ямами<sup>9</sup>. Гораздо проще использовать унарный предикат  $Pit$ , который принимает истинное значение в квадратах, содержащих ямы. Наконец, поскольку имеется точно один вампус, для его представления равным образом подходят и константа  $Wumpus$ , и унарный предикат (а последний способ обозначения с точки зрения вампуса может даже оказаться более почетным). Вампус проживает точно в одном квадрате, поэтому для именования этого квадрата целесообразно использовать функцию, такую как  $Home$  ( $Wumpus$ ). Это позволяет полностью избежать необходимости применения громоздкого множества высказываний, которые требовались в пропозициональной логике для обозначения того, что вампус находится точно в одном квадрате. (А при наличии двух вампусов ситуации в пропозициональной логике стала бы еще хуже.)

Местонахождение агента меняется со временем, поэтому мы будем применять запись  $At(Agent, s, t)$  для указания на то, что агент находится в квадрате  $s$  во время  $t$ . Зная свое текущее местонахождение, агент сможет выявлять путем логического вывода свойства текущего квадрата на основании данных о свойствах его текущего восприятия. Например, если агент находится в некотором квадрате и чувствует ветерок, то в этом квадрате чувствуется ветерок:

$$\forall s, t \ At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$$

Нам требуется знать не то, что агент вообще почувствовал ветерок, а то, что ветерок чувствуется в определенном квадрате, поскольку известно, что ямы не могут менять своего местонахождения. Обратите внимание на то, что предикат  $Breezy$  (“в квадрате чувствуется ветерок”) не имеет параметра с обозначением времени.

Обнаружив, в каких местах чувствуется ветерок (или неприятный запах), а также, что очень важно, в каких местах не чувствуется ветерок (или неприятный запах), агент получает возможность определять логическим путем, где находятся ямы (и где находится вампус). Существуют два описанных ниже типа синхронных правил, которые позволяют делать такие логические выводы.

-  **Диагностические правила**

Диагностические правила ведут от наблюдаемых эффектов к раскрытию скрытых причин. В очевидных диагностических правилах, касающихся поиска ям, утверждается, что если в квадрате чувствуется ветерок, то в некотором соседнем квадрате должна находиться яма, таким образом:

$$\forall s \ Breezy(s) \Rightarrow \exists r \ Adjacent(r, s) \wedge Pit(r)$$

С другой стороны, если в некотором квадрате не чувствуется ветерок, то ни в одном из соседних квадратов не находится яма<sup>10</sup>:

$$\forall s \ \neg Breezy(s) \Rightarrow \neg \exists r \ Adjacent(r, s) \wedge Pit(r)$$

---

<sup>9</sup> Аналогичным образом, большинство из нас не присваивают имя каждой птице, которая пролетает у нас над головой или переселяется в теплые края на зиму. С другой стороны, орнитологи, желающие изучить пути миграции, показатели выживания и т.д., присваивают имя каждой отловленной птице, закрепив кольцо на ее ноге, поскольку им приходится следить за отдельными птицами.

<sup>10</sup> Люди имеют естественную склонность забывать записывать отрицательную информацию, подобную этой. В обычной беседе такая склонность вполне оправдана. Было бы странно услышать слова: “На столе — две чашки, а не три или больше”, даже несмотря на то, что, формально говоря, утверждение: “На столе — две чашки” остается истинным, даже если на столе их три. Мы вернемся к этой теме в главе 10.

Объединяя эти два правила, мы получим следующее высказывание в форме двусторонней импликации:

$$\forall s \text{ Breezy}(s) \Leftrightarrow \exists r \text{ Adjacent}(r, s) \wedge \text{Pit}(r) \quad (8.3)$$

- **❖ Причинные правила**

Причинные правила отражают предполагаемую направленность причинно-следственных отношений в мире: появление некоторых восприятий вызывается определенными скрытыми свойствами мира. Например, наличие ямы вызывает появление ветерка во всех соседних квадратах:

$$\forall r \text{ Pit}(r) \Rightarrow [\forall s \text{ Adjacent}(r, s) \Rightarrow \text{Breezy}(s)]$$

А если во всех квадратах, соседних по отношению к данному конкретному квадрату, нет ям, то в данном квадрате не чувствуется ветерка:

$$\forall s [\forall r \text{ Adjacent}(r, s) \Rightarrow \neg \text{Pit}(r)] \Rightarrow \neg \text{Breezy}(s)$$

Приложив определенные усилия, можно показать, что эти два высказывания, вместе взятые, логически эквивалентны высказыванию с двусторонней импликацией, приведенному в уравнении 8.3. Это высказывание с двусторонней импликацией само может также рассматриваться как причинное, поскольку показывает, каким образом вырабатывается истинностное значение предиката *Breezy* по данным о состоянии мира.

Системы, в которых логические рассуждения формируются с помощью причинных правил, называются системами **❖ формирования рассуждений на основе модели**, поскольку причинные правила образуют модель того, как действует среда. Различие между рассуждениями на основе модели и диагностическими рассуждениями является важным для многих областей искусственного интеллекта. В частности, одной из активных областей исследования остается медицинская диагностика, и в ней подходы, основанные на прямых ассоциациях между симптомами и заболеваниями (диагностический подход), постепенно вытесняются подходами, в которых используются явно заданные модели заболеваний и данные о проявлениях этих заболеваний в виде симптомов. Мы вернемся к этой теме в главе 13.

Какой бы тип представления не использовался в агенте, **❖ если аксиомы правильно и полно описывают способ функционирования мира и способ выработки восприятий, то любая полная процедура логического вывода позволяет получить наиболее сильное возможное описание состояния мира при наличии всех доступных восприятий**. Благодаря этому проектировщик агента получает возможность сосредоточиться на решении задачи получения правильных знаний, не слишком заботясь о том, как должны быть организованы процессы логического вывода. Кроме того, в данном разделе было показано, что логика первого порядка позволяет представить мир намного более кратко, чем первоначальное описание на естественном языке, приведенное в главе 7.

## 8.4. ИНЖЕНЕРИЯ ЗНАНИЙ С ПРИМЕНЕНИЕМ ЛОГИКИ ПЕРВОГО ПОРЯДКА

В предыдущем разделе иллюстрировалось использование логики первого порядка для представления знаний в трех простых проблемных областях. В этом разделе рас-

сматривается общий процесс конструирования базы знаний, называемый **инженерией знаний**. *Инженером по знаниям* называется специалист, который исследует конкретную проблемную область, определяет, какие понятия важны в этой проблемной области, и создает формальное представление объектов и отношений в этой проблемной области. В данном разделе процесс инженерии знаний будет проиллюстрирован на проблемной области проектирования электронных схем, которая должна быть уже довольно знакомой читателю, поэтому мы можем сосредоточиться на самих относящихся к этой теме проблемах представления знаний. Применяемый в этом разделе подход является приемлемым для разработки баз знаний специального назначения, проблемная область которых тщательно очерчена и спектр запросов известен заранее. Базы знаний общего назначения, которые предназначены для поддержки запросов, касающихся полного спектра человеческих знаний, обсуждаются в главе 10.

## Процесс инженерии знаний

Безусловно, проекты в области инженерии знаний во многом отличаются друг от друга по своему содержанию, охвату и сложности, но все эти проекты включают перечисленные ниже этапы.

- 1. Идентификация задания.** Инженер по знаниям должен очертить круг вопросов, которые должна поддерживать база знаний, и виды фактов, которые будут доступными применительно к каждому конкретному экземпляру задачи. Например, должна ли база знаний о вампусе предоставлять возможность выбирать действия, или от нее требуется только поиск ответов на вопросы о содержании различных компонентов среды? Должны ли факты, полученные от датчиков, включать данные о текущем местонахождении? Само задание определяет, какие знания должны быть представлены в базе, чтобы можно было связать экземпляры задачи с ответами. Этот этап аналогичен процессу PEAS проектирования агентов, описанному в главе 2.
- 2. Сбор относящихся к делу знаний.** Инженер по знаниям может уже быть экспертом в рассматриваемой проблемной области или ему может потребоваться общаться с настоящими экспертами для выявления всего, что они знают — этот процесс называется **приобретением знаний**. На этом этапе знания еще не представлены формально. Его назначение состоит в том, чтобы понять, каким должен быть спектр знаний в базе знаний, определяемый самим заданием, а также разобраться в том, как фактически функционирует рассматриваемая проблемная область.  
Относящиеся к делу (релевантные) знания для мира вампуза, который определен с помощью искусственного набора правил, выявить несложно. (Но следует отметить, что определение понятия соседства квадратов не формулировалось явно в правилах функционирования мира вампуза.) Тем не менее для реальных проблемных областей задача выявления релевантных знаний может оказаться весьма сложной; например, система для эмуляции работы спроектированных СБИС может требовать или не требовать учета паразитных емкостей и поверхностных эффектов.
- 3. Определение словаря предикатов, функций и констант.** В иной формулировке этот этап можно определить как преобразование важных понятий уровня

проблемной области в имена логического уровня. Для этого необходимо ответить на многие вопросы в стиле инженерии знаний. Как и от стиля программирования, от стиля инженерии знаний может существенным образом зависеть окончательный успех проекта. Например, должны ли ямы быть представлены с помощью объектов или с помощью унарного предиката, определенного на квадратах? Должна ли ориентация агента быть задана в виде функции или предиката? Должно ли местонахождение вампуса зависеть от времени? Результатом выбора наиболее подходящих средств представления становится словарь, известный под названием **онтологии** проблемной области. Само слово **онтология** в данном контексте означает конкретную теорию пребывания в определенном состоянии, или теорию существования. Онтология определяет, какого рода объекты существуют, но не определяет их конкретные свойства и взаимосвязи.

- 4. Регистрация общих знаний о проблемной области.** Инженер по знаниям записывает аксиомы для всех терминов словаря. Тем самым он закрепляет (в возможной степени) смысл этих терминов, позволяя эксперту проверить их содержание. На этом этапе часто обнаруживаются неправильные трактовки или пропуски в словаре, который необходимо исправить, возвратившись на этап 3 и снова пройдя данную итерацию в текущем процессе проектирования.
- 5. Составление описания данного конкретного экземпляра задачи.** Если онтология хорошо продумана, этот этап будет несложным. Он сводится к написанию простых атомарных высказываний об экземплярах понятий, которые уже являются частью онтологии. Для логического агента определения экземпляров задачи поставляются датчиками, а сама “бестелесная” база знаний снабжается дополнительными высказываниями таким же образом, как традиционные программы снабжаются входными данными.
- 6. Передача запросов процедуре логического вывода и получение ответов.** Именно здесь нас ожидает награда: теперь мы можем применить процедуру логического вывода к аксиомам и фактам о конкретной задаче для получения фактов, которые нам хочется узнать.
- 7. Отладка базы знаний.** К сожалению, ответы на запросы при первой попытке редко оказываются правильными. Точнее, ответы будут правильными для базы знаний в том виде, в каком она написана, при условии, что процедура логического вывода является непротиворечивой, но они не будут такими, каких ожидает пользователь. Например, если недостает какой-то аксиомы, то на некоторые запросы из этой базы знаний нельзя будет найти ответ. В этом может помочь продуманный процесс отладки. Недостающие или слишком слабые аксиомы могут быть легко выявлены путем обнаружения участков, на которых неожиданно обрывается цепочка этапов логического вывода. Например, если база знаний содержит одну из диагностических аксиом, касающихся ям,

$$\forall s \text{ Breezy}(s) \Rightarrow \exists r \text{ Adjacent}(r, s) \wedge \text{Pit}(r)$$

но не содержит другую, то агент не сможет доказать отсутствие ям. Неправильные аксиомы могут быть выявлены на основании того, что они представляют собой ложные утверждения о мире. Например, аксиома

$$\forall x \text{ NumOfLegs}(x, 4) \Rightarrow \text{Mammal}(x)$$

является ложной, поскольку относит к млекопитающим рептилий, амфибий и, что еще важнее, столы с четырьмя ножками. *Ложность этого высказывания может быть определена независимо от остальной части базы знаний.* В отличие от этого, типичная ошибка в программе выглядит примерно таким образом:

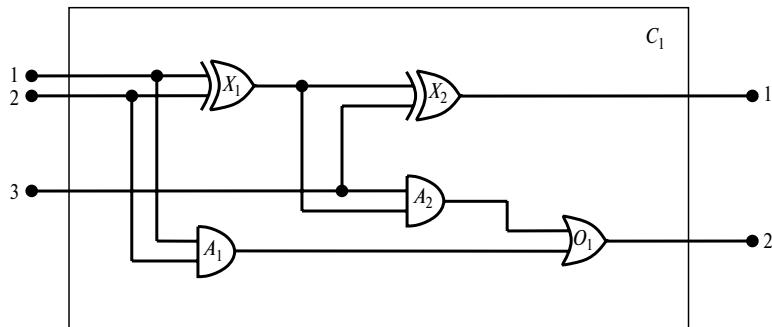
```
offset = position + 1
```

Невозможно определить, является ли этот оператор правильным, не изучив остальную часть программы для определения того, что, например, переменная `offset` используется для ссылки на текущую позицию или на позицию, которая следует за текущей позицией, а также происходит ли изменение значения переменной `position` в другом операторе и поэтому возникает необходимость снова изменять значение переменной `offset`.

Чтобы лучше понять этот процесс, состоящий из семи этапов, мы теперь применим его к расширенному примеру — к проблемной области электронных схем.

### Проблемная область электронных схем

Разработаем онтологию и базу знаний, которые позволяют нам рассуждать об электронных схемах такого типа, как показано на рис. 8.2. Мы будем руководствоваться описанным семиэтапным процессом инженерии знаний.



*Рис. 8.2. Цифровая схема  $C_1$ , предназначенная для использования в качестве однобитового полного сумматора. На первые два входа подаются два бита, подлежащие сложению, а на третий вход подается бит переноса. На первом выходе находится сумма, а на втором — бит переноса для следующего сумматора. Эта схема включает два логических элемента XOR, два логических элемента AND и один логический элемент OR*

### Идентификация задания

С цифровыми схемами связано много заданий, требующих логического рассуждения. На самом высоком уровне требуется проанализировать функциональное назначение схемы. Например, действительно ли приведенная на рис. 8.2 схема выполняет сложение должным образом? Если на все входы подаются сигналы с высоким потенциалом, то каким является выход логического элемента  $A_2$ ? Интерес представляют также вопросы о структуре схемы. Например, каковыми являются все логические элементы, подключенные к первой входной клемме? Содержит ли эта схема

петли обратной связи? В этом и состоят задания, рассматриваемые в данном разделе. Существуют также более детализированные уровни анализа, включая те, которые относятся к определению продолжительности задержек, площади схемы, потреблению энергии, стоимости производства и т.д. Для каждого из этих уровней могут потребоваться дополнительные знания.

### Сбор относящихся к делу знаний

Что мы знаем о цифровых схемах? Для наших целей достаточно знать, что они состоят из проводов и логических элементов. Сигналы распространяются по проводам к входным клеммам логических элементов, а каждый логический элемент выбирает на выходной клемме сигнал, который распространяется по другому проводу. Чтобы определить, какими должны быть эти сигналы, нам необходимо знать, как логические элементы преобразуют свои входные сигналы. Существует четыре основных типа логических элементов: логические элементы AND, OR и XOR имеют две входные клеммы, а логический элемент NOT имеет одну входную клемму. Все логические элементы имеют одну выходную клемму. Схемы, как и логические элементы, имеют входные и выходные клеммы.

Для того чтобы рассуждать о функциональных назначениях и связях в электронной схеме, не нужно вести речь о самих проводах, о путях, по которым проложены провода, или о соединениях, в которых встречаются два провода. Играют роль только соединения между клеммами — можно утверждать, что одна выходная клемма соединена с другой, входной клеммой, не упоминая о том, что они фактически соединены проводами. В этой проблемной области имеется также множество других факторов, не имеющих отношения к нашему анализу, таких как размеры, форма, цвет или стоимость различных компонентов.

Если бы нашей целью было нечто иное, чем проверка проектов на уровне логических элементов, то онтология была бы другой. Например, если бы нас интересовала отладка неисправных электронных схем, то, по-видимому, целесообразно было бы включить в онтологию провода, поскольку неисправный провод может исказить проходящий по нему сигнал. С другой стороны, для устранения ошибок синхронизации потребовалось бы включить в рассмотрение задержки логических элементов. А если бы мы были заинтересованы в проектировании продукта, который оказался бы прибыльным, то имели бы значение такие данные, как стоимость производства электронной схемы и ее быстродействие в сравнении с другими продуктами на рынке.

### Определение словаря

Теперь нам известно, что нужно вести речь о схемах, клеммах, сигналах и логических элементах. На следующем этапе необходимо выбрать функции, предикаты и константы для их представления. Начнем с отдельных логических элементов и перейдем к схемам.

Прежде всего необходимо иметь возможность отличать один логический элемент от других логических элементов. Это можно обеспечить, присвоив логическим элементам имена с помощью констант:  $X_1$ ,  $X_2$  и т.д. Хотя каждый логический элемент подключен к схеме своим индивидуальным способом, его поведение (способ преобразования входных сигналов в выходные) зависит только от типа. Для ссылки на тип

логического элемента может использоваться функция<sup>11</sup>. Например, можно написать  $Type(X_1) = \text{XOR}$ . Тем самым вводится константа XOR для логического элемента конкретного типа; другие константы будут называться OR, AND и NOT. Функция Type не является единственным способом регистрации онтологических различий. Для этого можно было бы использовать бинарный предикат,  $Type(X_1, \text{XOR})$ , или несколько отдельных предикатов типов, таких как  $\text{XOR}(X_1)$ . Любой из этих вариантов вполне подходит, но выбирая функцию Type, мы избегаем необходимости определять аксиому, в которой указано, что каждый отдельный логический элемент может иметь только один тип, поскольку это уже гарантирует сама семантика функций.

Затем рассмотрим клеммы. Логический элемент или цифровая схема может иметь одну или несколько входных клемм и одну или несколько выходных клемм. Можно было бы просто присвоить каждой из них имя с помощью константы, точно так же, как мы именовали логические элементы. Таким образом, логический элемент  $X_1$  мог бы иметь клеммы с именами  $X_1.In_1$ ,  $X_1.In_2$  и  $X_1.Out_1$ . Но следует избегать тенденции к созданию длинных составных имен. Назвав нечто  $X_1.In_1$ , мы не сделаем его первой входной клеммой  $X_1$ ; все равно потребуется указать это с помощью явного утверждения. По-видимому, лучше именовать клеммы логического элемента с помощью функции, так же, как мы именовали левую ногу короля Джона как  $LeftLeg(John)$ . Таким образом, примем, что  $In(1, X_1)$  обозначает первую входную клемму для логического элемента  $X_1$ . Аналогичная функция  $Out$  используется для выходных клемм.

Связь между логическими элементами может быть представлена с помощью предиката *Connected*, который принимает в качестве параметров имена двух клемм, как в выражении  $Connected(Out(1, X_1), In(1, X_2))$ .

Наконец, необходимо знать, включен или выключен сигнал. Одна из возможностей состоит в использовании унарного предиката *On*, который принимает истинное значение, когда сигнал на клемме включен. Тем не менее при этом затрудняется постановка таких вопросов, как: “Каковы возможные значения сигналов на выходных клеммах схемы  $C_1$ ?”. Поэтому введем в качестве объектов два “значения сигнала”, 1 и 0, и функцию *Signal*, которая принимает имя клеммы в качестве параметра и указывает значение сигнала для этой клеммы.

### **Регистрация общих знаний о проблемной области**

Одним из признаков наличия хорошей онтологии является то, что с ее использованием требуется определить очень немного общих правил, а признаком наличия хорошего словаря служит то, что каждое правило может быть сформулировано четко и кратко. В рассматриваемом примере нам потребуется только семь приведенных ниже простых правил для описания всего, что нужно знать о цифровых схемах.

1. Если две клеммы соединены, то на них присутствует один и тот же сигнал:

$$\forall t_1, t_2 \text{ } Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$$

---

<sup>11</sup> Обратите внимание на то, что в этом разделе имена, начинающиеся с соответствующих букв ( $A_{-1}$ ,  $X_{-1}$  и т.д.), используются исключительно для того, чтобы данный пример стал более удобным для чтения. База знаний все еще должна содержать информацию о типе для всех этих логических элементов.

2. Сигнал на каждой клемме равен либо 1, либо 0 (но не имеет оба значения одновременно):

$$\forall t \ Signal(t) = 1 \vee Signal(t) = 0 \\ 1 \neq 0$$

3. Предикат *Connected* является коммутативным:

$$\forall t_1, t_2 \ Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1)$$

На выходе логического элемента OR присутствует 1 тогда и только тогда, когда на любом из его входов присутствует 1:

$$\forall g \ Type(g) = OR \Rightarrow \\ Signal(Out(1, g)) = 1 \Leftrightarrow \forall n \ Signal(In(n, g)) = 1$$

На выходе логического элемента AND присутствует 0 тогда и только тогда, когда на любом из его входов присутствует 0:

$$\forall g \ Type(g) = AND \Rightarrow \\ Signal(Out(1, g)) = 0 \Leftrightarrow \forall n \ Signal(In(n, g)) = 0$$

На выходе логического элемента XOR присутствует 1 тогда и только тогда, когда на его входах присутствуют разные сигналы:

$$\forall g \ Type(g) = XOR \Rightarrow \\ Signal(Out(1, g)) = 1 \Leftrightarrow Signal(In(1, g)) \neq Signal(In(2, g))$$

Выход логического элемента NOT противоположен его входу:

$$\forall g \ (Type(g) = NOT) \Rightarrow Signal(Out(1, g)) \neq Signal(In(1, g))$$

### Составление конкретного экземпляра задачи

Схема, показанная на рис. 8.2, представлена как схема  $C_1$  со следующим описанием. Прежде всего определим типы всех логических элементов:

$$\begin{aligned} Type(X_1) &= XOR \\ Type(X_2) &= XOR \\ Type(A_1) &= AND \\ Type(A_2) &= AND \\ Type(O_1) &= OR \end{aligned}$$

Затем покажем связи между ними:

$$\begin{aligned} &Connected(Out(1, X_1), In(1, X_2)) \\ &Connected(In(1, C_1), In(1, X_1)) \\ &Connected(Out(1, X_1), In(2, A_2)) \\ &Connected(In(1, C_1), In(1, A_1)) \\ &Connected(Out(1, A_2), In(1, O_1)) \\ &Connected(In(2, C_1), In(2, X_1)) \\ &Connected(Out(1, A_1), In(2, O_1)) \\ &Connected(In(2, C_1), In(2, A_1)) \\ &Connected(Out(1, X_2), Out(1, C_1)) \\ &Connected(In(3, C_1), In(2, X_2)) \\ &Connected(Out(1, O_1), Out(2, C_1)) \\ &Connected(In(3, C_1), In(1, A_2)) \end{aligned}$$

### Передача запросов процедуре логического вывода

Какая комбинация входов вызовет появление 0 на первом выходе схемы  $C_1$  (бит суммы) и появление 1 на втором выходе схемы  $C_1$  (бит переноса)?

$$\begin{aligned} \exists i_1, i_2, i_3 \ Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge \\ Signal(In(3, C_1)) = i_3 \wedge Signal(Out(1, C_1)) = 0 \wedge \\ Signal(Out(2, C_1)) = 1 \end{aligned}$$

Ответами являются такие подстановки значений вместо переменных  $i_1$ ,  $i_2$  и  $i_3$ , что результирующее высказывание следует из базы знаний. Существуют три такие подстановки:

$$\{i_1/1, i_2/1, i_3/0\} \{i_1/1, i_2/0, i_3/1\} \{i_1/0, i_2/1, i_3/1\}$$

Каковы возможные множества значений сигналов на всех клеммах этой схемы сумматора?

$$\begin{aligned} \exists i_1, i_2, i_3, o_1, o_2 \ Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge \\ Signal(In(3, C_1)) = i_3 \wedge Signal(Out(1, C_1)) = o_1 \wedge \\ Signal(Out(2, C_1)) = o_2 \end{aligned}$$

Этот последний запрос должен вернуть полную таблицу входов и выходов для данного устройства, которая может использоваться для проверки того, действительно ли эта схема правильно складывает свои входные данные. Это — простой пример **проверки схемы**. Можно также применять приведенное здесь определение данной схемы для создания более крупных цифровых систем, для которых может осуществляться процедура проверки такого же рода (см. упр. 8.17). Для структурированной разработки базы знаний такого же рода подходят многие проблемные области, в которых более сложные понятия определяются на основе более простых понятий.

### Отладка базы знаний

Эту базу знаний можно испортить многими способами для определения того, какие виды ошибочного поведения будут с этим связаны. Например, допустим, что пропущено такое утверждение<sup>12</sup>, как  $1 \neq 0$ . Тогда система внезапно потеряет способность доказывать наличие каких-либо входных сигналов для данной схемы, за исключением тех случаев, когда на входе присутствуют сигналы 000 и 110. Наличие этой проблемы можно выявить, запрашивая выводы каждого логического элемента. Например, можно ввести следующий запрос:

$$\begin{aligned} \exists i_1, i_2, o \ Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge \\ Signal(Out(1, X_1)) = o \end{aligned}$$

который позволит обнаружить, что выходы  $X_1$  становятся неизвестными в случае входов 10 и 01. Затем мы рассмотрим аксиому для логических элементов XOR применительно к  $X_1$ :

$$Signal(Out(1, X_1)) = 1 \Leftrightarrow Signal(In(1, X_1)) \neq Signal(In(2, X_1))$$

Если известно, что входы, скажем, равны 1 и 0, то эта аксиома сводится к следующей:

---

<sup>12</sup> Такого рода упущения встречаются очень часто, поскольку люди обычно предполагают, что разные имена относятся к разным объектам. Аналогичное предположение принято также в системах логического программирования, которые рассматриваются в главе 9.

$$\text{Signal}(\text{Out}(1, X_1)) = 1 \Leftrightarrow 1 \neq 0$$

Теперь проблема становится очевидной: система неспособна получить заключение, что  $\text{Signal}(\text{Out}(1, X_1)) = 1$ , поэтому ей необходимо сообщить, что  $1 \neq 0$ .

## 8.5. РЕЗЮМЕ

---

В этой главе приведены вводные сведения о **логике первого порядка** — языке представления, который является гораздо более мощным по сравнению с пропозициональной логикой. Ниже перечислены наиболее важные понятия, представленные в настоящей главе.

- Языки представления знаний должны быть декларативными, композиционными, выразительными, независимыми от контекста и непротиворечивыми.
- Различные варианты логики отличаются друг от друга по своему **онтологическому вкладу** и **эпистемологическому вкладу**. Пропозициональная логика вносит вклад только в общий объем сведений, касающихся существования фактов, а логика первого порядка позволяет наращивать объем сведений, касающихся существования объектов и отношений, поэтому приобретает значительную выразительную мощь.
- **Возможный мир**, или **модель**, для логики первого порядка определяется множеством объектов, отношениями между ними и функциями, которые могут к ним применяться.
- **Константные символы** именуют объекты, **предикатные символы** именуют отношения, а **функциональные символы** именуют функции. **Интерпретация** задает отображение между символами и моделью. В **сложных термах** функциональные символы применяются к термам для именования объекта. Истинность высказывания определяется при наличии интерпретации и модели.
- **Атомарное высказывание** состоит из предиката, применяемого к одному или нескольким термам; оно становится истинным тогда и только тогда, когда имеет место отношение, обозначенное предикатом, между объектами, обозначенными его термами. В **сложных высказываниях**, так же как и в пропозициональной логике, используются связки, а **высказывания с кванторами** позволяют выражать общие правила.
- Для разработки базы знаний в логике первого порядка требуется тщательно провести процесс анализа проблемной области, выбора словаря и составления аксиом, необходимых для поддержки желаемых процедур логического вывода.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

---

Даже в логике Аристотеля предпринимались попытки применять обобщения к объектам, но датой рождения подлинной логики первого порядка стало появление книги Готтлоба Фреге *Begriffschrift* (“Система обозначения понятий”) [496], в которой были впервые введены кванторы. Предложенная в этой книге возможность вклады-

вать кванторы была большим шагом вперед, но Фреге использовал громоздкую систему обозначений. (Один из примеров таких обозначений приведен на первой странице обложки данной книги.) Современная система обозначений для логики первого порядка появилась в основном благодаря Джузеппе Пеано [1185], но ее семантика практически идентична семантике, предложенной в работе Фреге. Хотя сейчас это кажется довольно странным, но аксиомы Пеано в значительной степени обязаны своим появлением работам Грассмана [588] и Дедекинда [375].

Основным барьером на пути разработки логики первого порядка была полная концентрация усилий ученых на одноместных предикатах за счет исключения из рассмотрения многоместных реляционных предикатов. Такое сосредоточение на одноместных предикатах в логических системах было почти повсеместным, начиная от Аристотеля и включая Буля. Первая систематическая трактовка отношений была выполнена Августусом де Морганом [351], который цитировал следующий пример, чтобы показать, какого рода логические выводы не позволяет выполнять логика Аристотеля: “Все лошади — животные; поэтому голова лошади — это голова животного”. Такой логический вывод неосуществим в логике Аристотеля, поскольку в любом допустимом правиле, способном поддерживать этот логический вывод, вначале необходимо проанализировать данное высказывание с использованием двухместного предиката “ $x$  — голова  $y$ ”. Логика отношений была глубоко исследована Чарльзом Сандерсон Пирсом [1196], который также разработал логику первого порядка независимо от Фреге, хотя и немного позднее [1197].

Леопольд Лёвенхейм дал систематическую трактовку теории моделей для логики первого порядка в 1915 году [951]. В его статье рассматривался также символ равенства как неотъемлемая часть логики. Результаты Лёвенхайма были дополнительно расширены Торальфом Сколемом [1424]. Альфред Тарский [1490], [1491] дал явное определение понятий истинности и модельно-теоретического выполнения в логике первого порядка с использованием теории множеств.

Первым, кто предложил использовать логику первого порядка в качестве инструментального средства для создания систем искусственного интеллекта, был Маккарти [1009]. Перспективы искусственного интеллекта на основе логики значительно расширились в результате разработки Робинсоном [1298] резолюции — описанной в главе 9 полной процедуры вывода для логики первого порядка. Подход, получивший название *логицистского*, зародился в Стэнфордском университете. Корделл Грин [591], [592] разработал первую систему формирования рассуждений в логике первого порядка, QA3, что привело к первым попыткам создания логического робота в институте SRI [466]. Логика первого порядка была применена Зохаром Манна и Ричардом Валдингером [976] для формирования рассуждений о программах, а позднее — Майклом Генезеретом [536] для формирования рассуждений об электронных схемах. В Европе для использования в лингвистическом анализе [285] и для создания общих декларативных систем [849] было разработано логическое программирование (ограниченная форма проведения рассуждений в логике первого порядка). Кроме того, в ходе разработки проекта LCF (Logic for Computable Functions) в Эдинбурге [580] был внесен большой вклад в вычислительную логику. История этих разработок будет описана более подробно в главах 9 и 10.

Существует целый ряд хороших современных вводных учебников по логике первого порядка. Одним из наиболее удобных для чтения является [1254]. В [438] эта область знаний рассматривается в перспективе, в большей степени ориентирован-

ной на математику. В высшей степени формальная трактовка логики первого порядка, наряду с описанием более сложных тем логики, предоставлена в [94]. В [977] приведено удобное для чтения введение в логику с точки зрения компьютерных наук. В [515] дано чрезвычайно строгое математическое описание логики первого порядка, наряду со значительным объемом материала, который может использоваться в области автоматического формирования рассуждений. В книге *Logical Foundations of Artificial Intelligence* [537] приведено не только солидное введение в логику, но и первое систематическое описание логических агентов с восприятиями и действиями.

## УПРАЖНЕНИЯ

- 8.1.** *Логическая база знаний* представляет мир с использованием множества высказываний без явной структуры. **Аналогическое** представление, с другой стороны, имеет физическую структуру, которая непосредственно соответствует структуре представляемого объекта. Возьмите дорожную карту своей страны в качестве образца аналогического представления фактов о стране. Двухмерная структура карты соответствует двухмерной поверхности данного региона.
- а)** Приведите пять примеров символов на языке карты.
  - б)** *Явным высказыванием* называется высказывание, фактически формулируемое разработчиком представления. *Неявным высказыванием* называется высказывание, которое следует из явных высказываний в соответствии со свойствами аналогического представления. Приведите по три примера неявных и явных высказываний на языке карты.
  - в)** Приведите три примера сведений о физической структуре территории вашей страны, которые не могут быть представлены на языке карты.
  - г)** Приведите два примера фактов, которые гораздо легче выразить на языке карты, чем на языке логики первого порядка.
  - д)** Приведите еще два примера полезных аналогических представлений. Каковы преимущества и недостатки каждого из этих языков, логического и аналогического?
- 8.2.** Рассмотрите базу знаний, содержащую только два высказывания:  $P(a)$  и  $P(b)$ . Следует ли из этой базы знаний высказывание  $\forall x P(x)$ ? Объясните свой ответ в терминах моделей.
- 8.3.** Является ли допустимым высказывание  $\exists x, y x = y$ ? Объясните, почему.
- 8.4.** Напишите такое логическое высказывание, что каждый мир, в котором оно является истинным, содержит точно один объект.
- 8.5.** Рассмотрите словарь символов, который содержит  $c$  константных символов,  $p_k$  предикатных символов с арностью  $k$  каждый и  $f_k$  функциональных символов с арностью  $k$  каждый, где  $1 \leq k \leq A$ . Допустим, что размер проблемной области ограничен значением  $D$ . Для каждой конкретной комбинации “интерпретация/модель” каждый предикатный или функциональный символ отображается, соответственно, на отношение или функцию с той же арностью. Может быть принято предположение, что функции в этой модели допускают, чтобы некоторые входные кортежи не имели значения для этой

функции (т.е. значение было невидимым объектом). Выведите формулу определения количества возможных комбинаций “интерпретация/модель” для проблемной области с  $D$  элементами. Не учитывайте необходимость устранения избыточных комбинаций.

- 8.6.** Представьте приведенные ниже высказывания в логике первого порядка, используя совместимый словарь (который вы должны определить).
- a) Некоторые студенты участвовали в экзамене по французскому языку весной 2001 года.
  - b) Каждый студент, который участвует в экзамене по французскому языку, сдает его.
  - c) Только один студент участвовал в экзаменах по греческому языку весной 2001 года.
  - d) Лучшая оценка по греческому всегда выше чем лучшая оценка по французскому.
  - e) Каждый человек, который покупает страховой полис, умен.
  - f) Ни один человек не покупает дорогой страховой полис.
  - g) Существует агент, который продает страховой полис только людям, которые не застрахованы.
  - h) Есть некий парикмахер, который бреет всех мужчин в городе, которые не бреются сами.
  - i) Любой человек, рожденный в Великобритании, каждым из родителей которого является британский гражданин или британский резидент, является британским гражданином по рождению.
  - j) Любой человек, рожденный вне Великобритании, одним из родителей которого является британский гражданин по рождению, является британским гражданином по происхождению.
  - k) Политические деятели могут постоянно вводить некоторых людей в заблуждение, они также могут вводить в заблуждение всех людей некоторое время, но они не могут постоянно вводить в заблуждение всех людей.
- 8.7.** Представьте высказывание: “Все немцы говорят на одном и том же языке” в исчислении предикатов. Используйте предикат  $Speaks(x, 1)$ , который означает, что лицо  $x$  говорит на языке 1.
- 8.8.** Какая аксиома необходима, чтобы вывести логически факт  $Female(Laura)$ , если даны факты  $Male(Jim)$  и  $Spouse(Jim, Laura)$ ?
- 8.9.** Напишите общее множество фактов и аксиом, чтобы представить утверждение: “Веллингтон слышал о смерти Наполеона” и правильно ответить на вопрос: “Наполеон слышал о смерти Веллингтона?”
- 8.10.** Запишите в логике первого порядка факты о мире вампира, представленные в разделе 7.5 с помощью пропозициональной логики. Насколько более компактной является новая версия?
- 8.11.** Составьте аксиомы с описанием предикатов  $GrandChild$  (Внук или внучка),  $GreatGrandparent$  (Прадедушка или прабабушка),  $Brother$  (Брат),  $Sister$  (Сестра),  $Daughter$  (Дочь),  $Son$  (Сын),  $Aunt$  (Тетя),  $Uncle$  (Дядя),  $BrotherInLaw$  (Брат мужа или жены),  $SisterInLaw$  (Сестра мужа или жены) и

*FirstCousin* (Двоюродный брат или двоюродная сестра). Найдите правильное определение m-го кузена или кузины в n-м колене и запишите это определение в логике первого порядка.

Теперь запишите основные факты, изображенные в генеалогическом дереве, приведенном на рис. 8.3. С использованием подходящей системы формирования логических рассуждений введите в базу знаний с помощью операции *Tell* все записанные вами высказывания, а затем определите с помощью операции *Ask*, кто является внуками или внучками Элизабет (Elizabeth), братьями мужа Дианы (Diana), а также прадедушкой и прабабушкой Сары (Sarah).

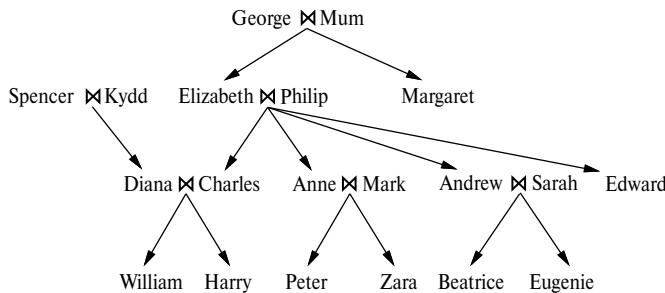


Рис. 8.3. Типичное генеалогическое дерево. Символ  $\bowtie$  соединяет супружеских, а стрелки указывают на детей

- 8.12. Запишите высказывание с утверждением, что функция  $+$  является коммутативной. Следует ли это высказывание из аксиом Пеано? В случае положительного ответа объясните, почему; в случае отрицательного ответа приведите модель, в которой эти аксиомы являются истинными, а ваше высказывание — ложным.
- 8.13. Объясните, в чем ошибка в следующем предлагаемом определении предиката принадлежности к множеству,  $\in$ :
$$\forall x, s \ x \in \{x | s\}$$

$$\forall x, s \ x \in s \Rightarrow \forall y \ x \in \{y | s\}$$
- 8.14. Используя в качестве примеров аксиомы множества, запишите аксиомы для проблемной области списков, включая все константы, функции и предикаты, упомянутые в данной главе.
- 8.15. Объясните, в чем ошибка в следующем предлагаемом определении соседних квадратов в мире вампуса:
$$\forall x, y \text{ } \textit{Adjacent}([x, y], [x+1, y]) \wedge \textit{Adjacent}([x, y], [x, y+1])$$
- 8.16. Запишите аксиомы, требуемые для формирования рассуждений о местонахождении вампуса, с использованием константного символа *Wumpus* и бинарного предиката *In* (*Wumpus*, *Location*). Не забудьте учсть, что существует только один вампус.
- 8.17. Дополните словарь, приведенный в разделе 8.4, чтобы определить правила сложения для n-битовых двоичных чисел. Затем сформулируйте описание четырехбитового сумматора, показанного на рис. 8.4, и составьте запросы, необходимые для проверки того, что это описание действительно правильно.

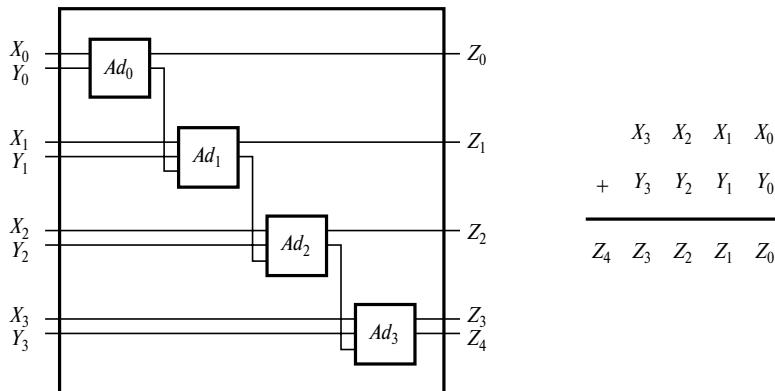


Рис. 8.4. Четырехбитовый сумматор

- 8.18.** Представление схемы, приведенное в этой главе, является более подробным, чем это необходимо, если нас интересуют только функциональные возможности этой схемы. В более простой формулировке рассматриваются любые логические элементы или цифровые схемы с  $m$  входами и  $n$  выходами с использованием предиката, имеющего  $m+n$  параметров, такого, что предикат принимает истинное значение тогда и только тогда, когда входы и выходы согласованы. Например, логические элементы NOT могут быть описаны с помощью бинарного предиката  $\text{NOT}(i, o)$ , для которого известны значения  $\text{NOT}(0, 1)$  и  $\text{NOT}(1, 0)$ . Композиции логических элементов определяются с помощью конъюнкций предикатов логических элементов, в которых наличие разделяемых переменных указывает на прямые соединения. Например, схема NAND может состоять из элементов AND и NOT:

$$\forall i_1, i_2, o_a, o \text{ NAND}(i_1, i_2, o) \Leftrightarrow \text{AND}(i_1, i_2, o_a) \wedge \text{NOT}(o_a, o)$$

С использованием этого представления определите однобитовый сумматор, приведенный на рис. 8.2, и четырехбитовый сумматор, приведенный на рис. 8.4, и объясните, какими запросами вы бы воспользовались для проверки таких проектов. Какого рода запросы, поддерживаемые представлением, описанным в разделе 8.4, не поддерживаются данным представлением?

- 8.19.** Получите бланк заявки на оформление паспорта для гражданина своей страны, выясните, какие правила определяют права человека на получение паспорта, и переведите эти правила на язык логики первого порядка в соответствии с этапами, описанными в разделе 8.4.

# 9 ЛОГИЧЕСКИЙ ВЫВОД В ЛОГИКЕ ПЕРВОГО ПОРЯДКА

*В этой главе будут определены эффективные процедуры получения ответов на вопросы, сформулированные в логике первого порядка.*

В главе 7 определено понятие **логического вывода** и показано, как можно обеспечить непротиворечивый и полный логический вывод для пропозициональной логики. В данной главе эти результаты будут дополнены для получения алгоритмов, позволяющих найти ответ на любой вопрос, сформулированный в логике первого порядка и имеющий ответ. Обладать такой возможностью очень важно, поскольку в логике первого порядка можно сформулировать практически любые знания, приложив для этого достаточные усилия.

В разделе 9.1 представлены правила логического вывода для кванторов и показано, как можно свести вывод в логике первого порядка к выводу в пропозициональной логике, хотя и за счет значительных издержек. В разделе 9.2 описана идея **унификации** и показано, как эта идея может использоваться для формирования правил логического вывода, которые могут применяться непосредственно к высказываниям в логике первого порядка. После этого рассматриваются три основных семейства алгоритмов вывода в логике первого порядка: **прямой логический вывод** и его применение к **дедуктивным базам данных** и **продукционным системам** рассматриваются в разделе 9.3; процедуры **обратного логического вывода** и системы **логического программирования** разрабатываются в разделе 9.4; а системы **доказательства теорем** на основе резолюции описаны в разделе 9.5. Вообще говоря, в любом случае следует использовать наиболее эффективный метод, позволяющий охватить все факты и аксиомы, которые должны быть выражены в процессе логического вывода. Но следует учитывать, что формирование рассуждений с помощью полностью общих высказываний логики первого порядка на основе метода резолюции обычно является менее эффективным по сравнению с формированием рассуждений с помощью определенных выражений с использованием прямого или обратного логического вывода.

## 9.1. СРАВНЕНИЕ МЕТОДОВ ЛОГИЧЕСКОГО ВЫВОДА В ПРОПОЗИЦИОНАЛЬНОЙ ЛОГИКЕ И ЛОГИКЕ ПЕРВОГО ПОРЯДКА

В этом и следующих разделах будут представлены идеи, лежащие в основе современных систем логического вывода. Начнем описание с некоторых простых правил логического вывода, которые могут применяться к высказываниям с кванторами для получения высказываний без кванторов. Эти правила естественным образом приводят к идею, что логический вывод в логике первого порядка может осуществляться путем преобразования высказываний в логике первого порядка, хранящихся в базе знаний, в высказывания, представленные в пропозициональной логике, и дальнейшего использования пропозиционального логического вывода, а о том, как выполнить этот вывод, нам уже известно из предыдущих глав. В следующем разделе указано одно очевидное сокращение, которое приводит к созданию методов логического вывода, позволяющих непосредственно манипулировать высказываниями в логике первого порядка.

### Правила логического вывода для кванторов

Начнем с кванторов всеобщности. Предположим, что база знаний содержит следующую стандартную аксиому, которая передает мысль, содержащуюся во многих сказках, что все жадные короли — злые:

$$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

В таком случае представляется вполне допустимым вывести из нее любое из следующих высказываний:

$$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$

$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

$$\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$$

...

Согласно правилу **конкретизации высказывания с квантором всеобщности** (сокращенно UI — Universal Instantiation), мы можем вывести логическим путем любое высказывание, полученное в результате подстановки **базового терма** (терма без переменных) вместо переменной, на которую распространяется квантор всеобщности<sup>1</sup>. Чтобы записать это правило логического вывода формально, воспользуемся понятием **подстановки**, введенным в разделе 8.3. Допустим, что  $\text{Subst}(\theta, \alpha)$  обозначает результат применения подстановки  $\theta$  к высказыванию  $\alpha$ . В таком случае данное правило для любой переменной  $v$  и базового терма  $g$  можно записать следующим образом:

$$\frac{\forall v \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

<sup>1</sup> Эти подстановки не следует путать с расширенными интерпретациями, которые использовались для определения семантики кванторов. В подстановке переменная заменяется термом (синтаксической конструкцией) для получения нового высказывания, тогда как любая интерпретация отображает некоторую переменную на объект в проблемной области.

Например, три высказывания, приведенные выше, получены с помощью подстановок  $\{x/John\}$ ,  $\{x/Richard\}$  и  $\{x/Father(John)\}$ .

Соответствующее правило  $\bowtie$  **конкретизации высказывания с квантором существования** (Existential Instantiation — EI) для квантора существования является немного более сложным. Для любых высказывания  $\alpha$ , переменной  $v$  и константного символа  $k$ , который не появляется где-либо в базе знаний, имеет место следующее:

$$\frac{\exists v \ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

Например, из высказывания

$$\exists x \ Crown(x) \wedge OnHead(x, John)$$

можно вывести высказывание

$$Crown(C_1) \wedge OnHead(C_1, John)$$

при условии, что константный символ  $C_1$  не появляется где-либо в базе знаний. По сути, в этом высказывании с квантором существования указано, что существует некоторый объект, удовлетворяющий определенному условию, а в процессе конкретизации просто присваивается имя этому объекту. Естественно, что это имя не должно уже принадлежать другому объекту. В математике есть прекрасный пример: предположим, мы открыли, что имеется некоторое число, которое немного больше чем 2,71828 и которое удовлетворяет уравнению  $d(x^y)/dy = x^y$  для  $x$ . Этому числу можно присвоить новое имя, такое как  $e$ , но было бы ошибкой присваивать ему имя существующего объекта, допустим,  $\pi$ . В логике такое новое имя называется  $\bowtie$  **сколемовской константой**. Конкретизация высказывания с квантором существования — это частный случай более общего процесса, называемого **сколемизацией**, который рассматривается в разделе 9.5.

Конкретизация высказывания с квантором существования не только сложнее, чем конкретизация высказывания с квантором всеобщности, но и играет в логическом выводе немного иную роль. Конкретизация высказывания с квантором всеобщности может применяться много раз для получения многих разных заключений, а конкретизация высказывания с квантором существования может применяться только один раз, а затем соответствующее высказывание с квантором существования может быть отброшено. Например, после того как в базу знаний будет добавлено высказывание  $Kill(Murderer, Victim)$ , становится больше не нужным высказывание  $\exists x Kill(x, Victim)$ . Строго говоря, новая база знаний логически не эквивалентна старой, но можно показать, что она  $\bowtie$  **эквивалентна с точки зрения логического вывода**, в том смысле, что она выполнима тогда и только тогда, когда выполнима первоначальная база знаний.

### Приведение к пропозициональному логическому выводу

Получив в свое распоряжение правила вывода высказываний с кванторами из высказываний без кванторов, мы получаем возможность привести вывод в логике первого порядка к выводу в пропозициональной логике. В данном разделе будут изложены основные идеи этого процесса, а более подробные сведения приведены в разделе 9.5.

Основная идея состоит в следующем: по аналогии с тем, как высказывание с квантором существования может быть заменено одной конкретизацией, высказы-

вание с квантором всеобщности может быть заменено множеством всех возможных конкретизаций. Например, предположим, что наша база знаний содержит только такие высказывания:

$$\begin{aligned} \forall x \ King(x) \wedge Greedy(x) &\Rightarrow Evil(x) \\ King(John) \\ Greedy(John) \\ Brother(Richard, John) \end{aligned} \tag{9.1}$$

Затем применим правило конкретизации высказывания с квантором всеобщности к первому высказыванию, используя все возможные подстановки базовых термов из словаря этой базы знаний — в данном случае  $\{x/John\}$  и  $\{x/Richard\}$ . Мы получим следующие высказывания:

$$\begin{aligned} King(John) \wedge Greedy(John) &\Rightarrow Evil(John) \\ King(Richard) \wedge Greedy(Richard) &\Rightarrow Evil(Richard) \end{aligned}$$

и отбросим высказывание с квантором всеобщности. Теперь база знаний становится по сути пропозициональной, если базовые атомарные высказывания ( $King(John)$ ,  $Greedy(John)$  и т.д.) рассматриваются как пропозициональные символы. Поэтому теперь можно применить любой из алгоритмов полного пропозиционального вывода из главы 7 для получения таких заключений, как  $Evil(John)$ .

Как показано в разделе 9.5, такой метод **пропозиционализации** (преобразования в высказывания пропозициональной логики) может стать полностью обобщенным; это означает, что любую базу знаний и любой запрос в логике первого порядка можно пропозиционализировать таким образом, чтобы сохранялось логическое следствие. Таким образом, имеется полная процедура принятия решения в отношении того, сохраняется ли логическое следствие... или, возможно, такой процедуры нет. Дело в том, что существует такая проблема: если база знаний включает функциональный символ, то множество возможных подстановок базовых термов становится бесконечным! Например, если в базе знаний упоминается символ  $Father$ , то существует возможность сформировать бесконечно большое количество вложенных термов, таких как  $Father(Father(Father(John)))$ . А применяемые нами пропозициональные алгоритмы сталкиваются с затруднениями при обработке бесконечно большого множества высказываний.

К счастью, имеется знаменитая теорема, предложенная Жаком Эрбраном [650], согласно которой, если некоторое высказывание следует из первоначальной базы знаний в логике первого порядка, то существует доказательство, которое включает лишь конечное подмножество этой пропозиционализированной базы знаний. Поскольку любое такое подмножество имеет максимальную глубину вложения среди его базовых термов, это подмножество можно найти, формируя вначале все конкретизации с константными символами ( $Richard$  и  $John$ ), затем все термы с глубиной 1 ( $Father(Richard)$  и  $Father(John)$ ), после этого все термы с глубиной 2 и т.д. до тех пор, пока мы не сможем больше составить пропозициональное доказательство высказывания, которое следует из базы знаний.

Выше был кратко описан один из подходов к организации вывода в логике первого порядка с помощью пропозиционализации, который является **полным**, т.е. позволяет доказать любое высказывание, которое следует из базы знаний. Это — важное достижение, если учесть, что пространство возможных моделей является бесконечным. С другой стороны, до тех пор пока это доказательство не составлено, мы не

знаем, следует ли данное высказывание из базы знаний! Что произойдет, если это высказывание из нее не следует? Можем ли мы это определить? Как оказалось, для логики первого порядка это действительно невозможно. Наша процедура доказательства может продолжаться и продолжаться, вырабатывая все более и более глубоко вложенные термы, а мы не будем знать, вошла ли она в безнадежный цикл или до получения доказательства остался только один шаг. Такая проблема весьма напоминает проблему останова машин Тьюринга. Алан Тьюринг [1518] и Алонсо Черч [255] доказали неизбежность такого состояния дел, хотя и весьма различными способами.

*Вопрос о следствии для логики первого порядка является полуразрешимым; это означает, что существуют алгоритмы, которые позволяют найти доказательство для любого высказывания, которое следует из базы знаний, но нет таких алгоритмов, которые позволяли бы также определить, что не существует доказательства для каждого высказывания, которое не следует из базы знаний.*

## 9.2. УНИФИКАЦИЯ И ПОДНЯТИЕ

В предыдущем разделе описан уровень понимания процесса вывода в логике первого порядка, который существовал вплоть до начала 1960-х годов. Внимательный читатель (и, безусловно, специалисты в области вычислительной логики, работавшие в начале 1960-х годов) должен был заметить, что подход на основе пропозиционализации является довольно неэффективным. Например, если заданы запрос  $Evil(x)$  и база знаний, приведенная в уравнении 9.1, то становится просто нерациональным формирование таких высказываний, как  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$ . И действительно, для любого человека вывод факта  $Evil(John)$  из следующих высказываний кажется вполне очевидным:

$$\begin{aligned} \forall x \ King(x) \wedge Greedy(x) &\Rightarrow Evil(x) \\ King(John) \\ Greedy(John) \end{aligned}$$

Теперь мы покажем, как сделать его полностью очевидным для компьютера.

### Правило вывода в логике первого порядка

Процедура вывода того факта, что Джон — злой, действует следующим образом: найти некоторый  $x$ , такой, что  $x$  — король и  $x$  — жадный, а затем вывести, что  $x$  — злой. Вообще говоря, если существует некоторая подстановка  $\theta$ , позволяющая сделать предпосылку импликации идентичной высказываниям, которые уже находятся в базе знаний, то можно утверждать об истинности заключения этой импликации после применения  $\theta$ . В данном случае такой цели достигает подстановка  $\{x/John\}$ .

Фактически можно обеспечить выполнение на этом этапе вывода еще больше работы. Предположим, что нам известно не то, что жаден Джон —  $Greedy(John)$ , а что жадными являются все:

$$\forall y \ Greedy(y) \tag{9.2}$$

Но и в таком случае нам все равно хотелось бы иметь возможность получить заключение, что Джон зол —  $Evil(John)$ , поскольку нам известно, что Джон — ко-

роль (это дано) и Джон жаден (так как жадными являются все). Для того чтобы такой метод мог работать, нам нужно найти подстановку как для переменных в высказывании с импликацией, так и для переменных в высказываниях, которые должны быть согласованы. В данном случае в результате применения подстановки  $\{x/John, y/John\}$  к предпосылкам импликации  $King(x)$  и  $Greedy(x)$  и к высказываниям из базы знаний  $King(John)$  и  $Greedy(y)$  эти высказывания становятся идентичными. Таким образом, теперь можно вывести заключение импликации.

Такой процесс логического вывода может быть представлен с помощью единственного правила логического вывода, которое будет именоваться **обобщенным правилом отделения** (Generalized Modus Ponens): для атомарных высказываний  $p_i$ ,  $p_i'$  и  $q$ , если существует подстановка  $\theta$ , такая, что  $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$ , то для всех  $i$  имеет место следующее:

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{Subst}(\theta, q)}$$

В этом правиле имеется  $n+1$  предпосылка:  $n$  атомарных высказываний  $p_i'$  и одна импликация. Заключение становится результатом применения подстановки  $\theta$  к следствию  $q$ . В данном примере имеет место следующее:

$$\begin{array}{ll} p_1' - \text{это } King(John) & p_1 - \text{это } King(x) \\ p_2' - \text{это } Greedy(y) & p_2 - \text{это } Greedy(x) \\ \theta - \text{это } \{x/John, y/John\} & q - \text{это } Evil(x) \\ \text{Subst}(\theta, q) - \text{это } Evil(John) & \end{array}$$

Можно легко показать, что обобщенное правило отделения — непротиворечивое правило логического вывода. Прежде всего отметим, что для любого высказывания  $p$  (в отношении которого предполагается, что на его переменные распространяется квантор всеобщности) и для любой подстановки  $\theta$  справедливо следующее правило:

$$p \models \text{Subst}(\theta, p)$$

Это правило выполняется по тем же причинам, по которым выполняется правило конкретизации высказывания с квантором всеобщности. Оно выполняется, в частности, в любой подстановке  $\theta$ , которая удовлетворяет условиям обобщенного правила отделения. Поэтому из  $p_1', \dots, p_n'$  можно вывести следующее:

$$\text{Subst}(\theta, p_1') \wedge \dots \wedge \text{Subst}(\theta, p_n')$$

а из импликации  $p_1 \wedge \dots \wedge p_n \Rightarrow q$  — следующее:

$$\text{Subst}(\theta, p_1) \wedge \dots \wedge \text{Subst}(\theta, p_n) \Rightarrow \text{Subst}(\theta, q)$$

Теперь подстановка  $\theta$  в обобщенном правиле отделения определена так, что  $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$  для всех  $i$ , поэтому первое из этих двух высказываний точно совпадает с предпосылкой второго высказывания. Таким образом, выражение  $\text{Subst}(\theta, q)$  следует из правила отделения.

Как принято выражаться в логике, обобщенное правило отделения представляет собой **поднятую** версию правила отделения — оно поднимает правило отделения из пропозициональной логики в логику первого порядка. В оставшейся части этой главы будет показано, что могут быть разработаны поднятые версии алгоритмов прямого логического вывода, обратного логического вывода и резолюции, представленных в главе 7. Основным преимуществом применения поднятых правил логиче-

ского вывода по сравнению с пропозиционализацией является то, что в них предусмотрены только те подстановки, которые требуются для обеспечения дальнейшего выполнения конкретных логических выводов. Единственное соображение, которое может вызвать недоумение у читателя, состоит в том, что в определенном смысле обобщенное правило вывода является менее общим, чем исходное правило отделения (с. 303): правило отделения допускает применение в левой части импликации любого отдельно взятого высказывания  $\alpha$ , а обобщенное правило отделения требует, чтобы это высказывание имело специальный формат. Но оно является обобщенным в том смысле, что допускает применение любого количества выражений  $p_i'$ .

## Унификация

Применение поднятых правил логического вывода связано с необходимостью поиска подстановок, в результате которых различные логические выражения становятся идентичными. Этот процесс называется **унификацией** и является ключевым компонентом любых алгоритмов вывода в логике первого порядка. Алгоритм *Unify* принимает на входе два высказывания и возвращает для них **унификатор**, если такой существует:

$$\text{Unify}(p, q) = \theta \text{ где } \text{Subst}(\theta, p) = \text{Subst}(\theta, q)$$

Рассмотрим несколько примеров того, как должен действовать алгоритм *Unify*. Предположим, что имеется запрос *Knows(John, x)* — кого знает Джон? Некоторые ответы на этот запрос можно найти, отыскивая все высказывания в базе знаний, которые унифицируются с высказыванием *Knows(John, x)*. Ниже приведены результаты унификации с четырьмя различными высказываниями, которые могут находиться в базе знаний.

$$\begin{aligned}\text{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(\text{John}, \text{Jane})) &= \{x/\text{Jane}\} \\ \text{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Bill})) &= \{x/\text{Bill}, y/\text{John}\} \\ \text{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Mother}(y))) &= \{y/\text{John}, x/\text{Mother}(\text{John})\} \\ \text{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{Elizabeth})) &= \text{fail}\end{aligned}$$

Последняя попытка унификации оканчивается неудачей (*fail*), поскольку переменная  $x$  не может одновременно принимать значения *John* и *Elizabeth*. Теперь вспомним, что высказывание *Knows(x, Elizabeth)* означает “Все знают Элизабет”, поэтому мы обязаны иметь возможность вывести логически, что Джон знает Элизабет. Проблема возникает только потому, что в этих двух высказываниях, как оказалось, используется одно и то же имя переменной,  $x$ . Возникновения этой проблемы можно избежать, **стандартизируя отличие** (*standardizing apart*) одного из этих двух унифицируемых высказываний; под этой операцией подразумевается переименование переменных в высказываниях для предотвращения коллизий имен. Например, переменную  $x$  в высказывании *Knows(x, Elizabeth)* можно переименовать в  $z_{17}$  (новое имя переменной), не меняя смысла этого высказывания. После этого унификация выполняется успешно:

$$\text{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(z_{17}, \text{Elizabeth})) = \{x/\text{Elizabeth}, z_{17}/\text{John}\}$$

С дополнительными сведениями о том, с чем связана необходимость в стандартизации отличия, можно ознакомиться в упр. 9.7.

Возникает еще одна сложность: выше было сказано, что алгоритм `Unify` должен возвращать такую подстановку (или унификатор), в результате которой два параметра становятся одинаковыми. Но количество таких унификаторов может быть больше единицы. Например, вызов алгоритма `Unify(Knows(John, x), Knows(y, z))` может возвратить  $\{y/John, x/z\}$  или  $\{y/John, x/John, z/John\}$ . Первый унификатор позволяет получить в качестве результата унификации выражение  $Knows(John, z)$ , а второй дает  $Knows(John, John)$ . Но второй результат может быть получен из первого с помощью дополнительной подстановки  $\{z/John\}$ ; в таком случае принято считать, что первый унификатор является более общим по сравнению со вторым, поскольку налагает меньше ограничений на значения переменных. Как оказалось, для любой унифицируемой пары выражений существует единственный **наиболее общий унификатор** (Most General Unifier — MGU), который является уникальным вплоть до переименования переменных. В данном случае таким является  $\{y/John, x/z\}$ .

Алгоритм вычисления наиболее общих унификаторов приведен в листинге 9.1. Процесс его работы очень прост: рекурсивно исследовать два выражения одновременно, “бок о бок”, наряду с этим формируя унификатор, но создавать ситуацию неудачного завершения, если две соответствующие точки в полученных таким образом структурах не совпадают. При этом существует один долгостоящий этап: если переменная согласуется со сложным термом, необходимо провести проверку того, встречается ли сама эта переменная внутри терма; в случае положительного ответа на данный вопрос согласование оканчивается неудачей, поскольку невозможно сформировать какой-либо совместимый унификатор. Из-за этой так называемой **проверки вхождения** (occig check) сложность всего алгоритма становится квадратично зависимой от размера унифицируемых выражений. В некоторых системах, включая все системы логического программирования, просто исключается такая проверка вхождения и поэтому в результате иногда формируются противоречивые логические выводы, а в других системах используются более развитые алгоритмы со сложностью, линейно зависящей от времени.

**Листинг 9.1. Алгоритм унификации.** Алгоритм действует путем поэлементного сравнения структур входных высказываний. В ходе этого формируется подстановка  $\theta$ , которая также является параметром функции `Unify` и используется для проверки того, что дальнейшие сравнения совместимы со связываниями, которые были определены ранее. В составном выражении, таком как  $F(A, B)$ , функция `Op` выбирает функциональный символ  $F$ , а функция `Args` выбирает список параметров ( $A, B$ )

---

```

function Unify(x, y, θ) returns подстановка, позволяющая сделать
    x и y идентичными
inputs: x, переменная, константа, список или составной терм
        y, переменная, константа, список или составной терм
        θ, подстановка, подготовленная до сих пор (необязательный
            параметр, по умолчанию применяется пустой терм)

    if θ = failure then return failure
    else if x = y then return θ
    else if Variable?(x) then return Unify-Var(x, y, θ)
    else if Variable?(y) then return Unify-Var(y, x, θ)
    else if Compound?(x) and Compound?(y) then

```

```

    return Unify(Args[x], Args[y], Unify(Op[x], Op[y], θ))
else if List?(x) and List?(y) then
    return Unify(Rest[x], Rest[y], Unify(First[x], First[y], θ))
else return failure

function Unify-Var(var, x, θ) returns подстановка
    inputs: var, переменная
            x, любое выражение
            θ, подстановка, подготовленная до сих пор

    if {var/val} ∈ θ then return Unify(val, x, θ)
    else if {x/val} ∈ θ then return Unify(var, val, θ)
    else if Occur-Check?(var, x) then return failure
    else return добавление {var/x} к подстановке θ

```

---

## Хранение и выборка

В основе функций `Tell` и `Ask`, применяемых для ввода информации и передачи запросов в базу знаний, лежат более примитивные функции `Store` и `Fetch`. Функция `Store(s)` сохраняет некоторое высказывание  $s$  в базе знаний, а функция `Fetch(q)` возвращает все унифициаторы, такие, что запрос  $q$  унифицируется с некоторым высказыванием из базы знаний. Описанная выше задача, служившая для иллюстрации процесса унификации (поиск всех фактов, которые унифицируются с высказыванием  $Knows(John, x)$ ), представляет собой пример применения функции `Fetch`.

Проще всего можно реализовать функции `Store` и `Fetch`, предусмотрев хранение всех фактов базы знаний в виде одного длинного списка, чтобы затем, после получения запроса  $q$ , можно было просто вызывать алгоритм `Unify(q, s)` для каждого высказывания  $s$  в списке. Такой процесс является неэффективным, но он осуществим, и знать об этом — это все, что нужно для понимания последней части данной главы. А в оставшейся части данного раздела описаны способы, позволяющие обеспечить более эффективную выборку, и он может быть пропущен при первом чтении.

Функцию `Fetch` можно сделать более эффективной, обеспечив, чтобы попытки унификации применялись только к высказываниям, имеющим определенный шанс на унификацию. Например, нет смысла пытаться унифицировать  $Knows(John, x)$  и  $Brother(Richard, John)$ . Такой унификации можно избежать, **индексируя** факты в базе знаний. Самая простая схема, называемая **индексацией по предикатам**, предусматривает размещение всех фактов `Knows` в одном сегменте, а всех фактов `Brother` — в другом. Сами сегменты для повышения эффективности доступа можно хранить в хэш-таблице<sup>2</sup>.

Индексация по предикатам является удобной, когда имеется очень много предикатных символов, но лишь небольшое количество выражений в расчете на каждый символ. Однако в некоторых приложениях имеется много выражений в расчете на

---

<sup>2</sup> Хэш-таблица — эта структура данных для хранения и выборки информации, индексируемой с помощью фиксированных ключей. С точки зрения практики хэш-таблица может рассматриваться как имеющая постоянные показатели хранения и выборки, даже если эта таблица содержит очень большое количество элементов.

каждый конкретный предикатный символ. Например, предположим, что налоговые органы желают следить за тем, кто кого нанимает, с использованием предиката  $Employs(x, y)$ . Такой сегмент, возможно, состоящий из миллионов нанимателей и десятков миллионов наемных работников, был бы очень большим. Для поиска ответа на такой запрос, как  $Employs(x, Richard)$  (“Кто является нанимателем Ричарда?”), при использовании индексации по предикатам потребовался бы просмотр всего сегмента.

Поиск ответа на данный конкретный запрос стал бы проще при использовании индексации фактов и по предикату, и по второму параметру, возможно, с использованием комбинированного ключа хэш-таблицы. В таком случае существовала бы возможность просто формировать ключ из запроса и осуществлять выборку именно тех фактов, которые унифицируются с этим запросом. А для ответа на другие запросы, такие как  $Employs(AIMA.org, y)$ , нужно было бы индексировать факты, комбинируя предикат с первым параметром. Поэтому факты могут храниться под разными индексными ключами, что позволяет моментально сделать их доступными для разных запросов, с которыми они могли бы унифицироваться.

Если дано некоторое высказывание, которое подлежит хранению, то появляется возможность сформировать индексы для всех возможных запросов, которые унифицируются с ними. Применительно к факту  $Employs(AIMA.org, Richard)$  возможны следующие запросы:

$Employs(AIMA.org, Richard)$	Является ли организация AIMA.org нанимателем Ричарда?
$Employs(x, Richard)$	Кто является нанимателем Ричарда?
$Employs(AIMA.org, y)$	Для кого является нанимателем организация AIMA.org?
$Employs(x, y)$	Кто для кого является нанимателем?

Как показано на рис. 9.1, а, эти запросы образуют **решетку обобщения**. Такая решетка обладает некоторыми интересными свойствами. Например, дочерний узел любого узла в этой решетке может быть получен из его родительского узла с помощью единственной подстановки, а “наибольший” общий потомок любых двух узлов является результатом применения наиболее общего унификатора для этих узлов. Та часть решетки, которая находится выше любого базового факта, может быть сформирована систематически (упр. 9.5). Высказывание с повторяющимися константами имеет несколько иную решетку, как показано на рис. 9.1, б. Наличие функциональных символов и переменных в высказываниях, подлежащих хранению, приводит к появлению еще более интересных структур решетки.

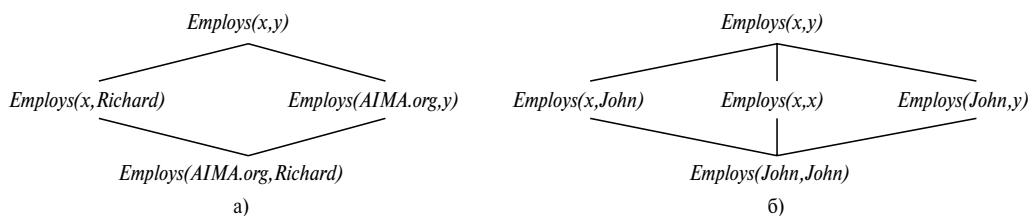


Рис. 9.1. Примеры решеток обобщения: решетка обобщения, самым нижним узлом которой является высказывание  $Employs(AIMA.org, Richard)$  (а); решетка обобщения для высказывания  $Employs(John, John)$  (б)

Только что описанная схема применяется очень успешно, если решетка содержит небольшое количество узлов. А если предикат имеет  $n$  параметров, то решетка включает  $O(2^n)$  узлов. Если же разрешено применение функциональных символов, то количество узлов также становится экспоненциально зависимым от размера термов в высказывании, подлежащем хранению. Это может вызвать необходимость создания огромного количества индексов. В какой-то момент затраты на хранение и сопровождение всех этих индексов перевесят преимущества индексации. Для выхода из этой ситуации можно применять какой-либо жесткий подход, например сопровождать индексы только на ключах, состоящих из некоторого предиката плюс каждый параметр, или адаптивный подход, в котором предусматривается создание индексов в соответствии с потребностями в поиске ответов на запросы того типа, которые встречаются наиболее часто. В большинстве систем искусственного интеллекта количество фактов, подлежащих хранению, является достаточно небольшим для того, чтобы проблему эффективной индексации можно было считать решенной. А что касается промышленных и коммерческих баз данных, то эта проблема стала предметом значительных и продуктивных технологических разработок.

### **9.3. ПРЯМОЙ ЛОГИЧЕСКИЙ ВЫВОД**

---

Алгоритм прямого логического вывода для пропозициональных определенных выражений приведен в разделе 7.5. Его идея проста: начать с атомарных высказываний в базе знаний и применять правило отделения в прямом направлении, добавляя все новые и новые атомарные высказывания до тех пор, пока не возникнет ситуация, в которой невозможно будет продолжать формулировать логические выводы. В данном разделе приведено описание того, как можно применить этот алгоритм к определенным выражениям в логике первого порядка и каким образом он может быть реализован эффективно. Определенные выражения, такие как *Situation*  $\Rightarrow$  *Response*, особенно полезны для систем, в которых логический вывод осуществляется в ответ на вновь поступающую информацию. Таким образом могут быть определены многие системы, а формирование рассуждений с помощью прямого логического вывода может оказаться гораздо более эффективным по сравнению с доказательством теорем с помощью резолюции. Поэтому часто имеет смысл попытаться сформировать базу знаний с использованием только определенных выражений, чтобы избежать издержек, связанных с резолюцией.

#### **Определенные выражения в логике первого порядка**

Определенные выражения в логике первого порядка весьма напоминают определенные выражения в пропозициональной логике (с. 312): они представляют собой дизъюнкцию литералов, среди которых положительным является один и только один. Определенное выражение либо является атомарным, либо представляет собой импликацию, антецедентом (предпосылкой) которой служит конъюнкция положительных литералов, а консеквентом (следствием) — единственный положительный литерал. Ниже приведены примеры определенных выражений в логике первого порядка.

$$\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

$$\text{King}(\text{John})$$

$$\text{Greedy}(y)$$

В отличие от пропозициональных литералов, литералы первого порядка могут включать переменные, и в таком случае предполагается, что на эти переменные распространяется квантор всеобщности. (Как правило, при написании определенных выражений кванторы всеобщности исключаются.) Определенные выражения представляют собой подходящую нормальную форму для использования в обобщенном правиле отделения.

Не все базы знаний могут быть преобразованы в множество определенных выражений из-за того ограничения, что положительный литерал в них должен быть единственным, но для многих баз знаний такая возможность существует. Рассмотрим приведенную ниже задачу.

Закон гласит, что продажа оружия недружественным странам, осуществляемая любым американским гражданином, считается преступлением. В государстве Ноуноу, враждебном по отношению к Америке, имеются некоторые ракеты, и все ракеты этого государства были проданы ему полковнику Уэстом, который является американским гражданином.

Мы должны доказать, что полковник Уэст совершил преступление. Вначале все имеющиеся факты будут представлены в виде определенных выражений в логике первого порядка, а в следующем разделе будет показано, как решить эту задачу с помощью алгоритма прямого логического вывода.

- “...продажа оружия враждебным странам, осуществляемая любым американским гражданином, является преступлением”:

$$\begin{aligned} \text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \\ \Rightarrow \text{Criminal}(x) \end{aligned} \quad (9.3)$$

- “В государстве Ноуноу... имеются некоторые ракеты”. Высказывание  $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$  преобразуется в два определенных выражения путем устранения квантора существования и введения новой константы  $M_1$ :

$$\text{Owns}(\text{Nono}, M_1) \quad (9.4)$$

$$\text{Missile}(M_1) \quad (9.5)$$

- “...все ракеты этого государства были проданы ему полковнику Уэстом”:

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \quad (9.6)$$

Нам необходимо также знать, что ракеты — оружие:

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x) \quad (9.7)$$

Кроме того, мы должны знать, что государство, враждебное по отношению к Америке, рассматривается как “недружественное”:

$$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \quad (9.8)$$

- “...полковнику Уэстом, который является американским гражданином”:

$$\text{American}(\text{West}) \quad (9.9)$$

- “В государстве Ноуноу, враждебном по отношению к Америке...”:

$$\text{Enemy}(\text{Nono}, \text{America}) \quad (9.10)$$

Эта база знаний не содержит функциональных символов и поэтому может служить примером класса баз знаний языка **Datalog**, т.е. примером множества определенных выражений в логике первого порядка без функциональных символов. Ниже будет показано, что при отсутствии функциональных символов логический вывод становится намного проще.

### Простой алгоритм прямого логического вывода

Как показано в листинге 9.2, первый рассматриваемый нами алгоритм прямого логического вывода является очень простым. Начиная с известных фактов, он активизирует все правила, предпосылки которых выполняются, и добавляет заключения этих правил к известным фактам. Этот процесс продолжается до тех пор, пока не обнаруживается ответ на запрос (при условии, что требуется только один ответ) или больше не происходит добавление новых фактов. Следует отметить, что факт не является “новым”, если он представляет собой **переименование** известного факта. Одно высказывание называется переименованием другого, если оба эти высказывания идентичны во всем, за исключением имен переменных. Например, высказывания *Likes(x, IceCream)* и *Likes(y, IceCream)* представляют собой переименования по отношению друг к другу, поскольку они отличаются лишь выбором имени переменной, *x* или *y*; они имеют одинаковый смысл — все любят мороженое.

**Листинг 9.2.** Концептуально простой, но очень неэффективный алгоритм прямого логического вывода. В каждой итерации он добавляет к базе знаний *KB* все атомарные высказывания, которые могут быть выведены за один этап из импликационных высказываний и атомарных высказываний, которые уже находятся в базе знаний

---

```

function FOL-FC-Ask(KB,  $\alpha$ ) returns подстановка или значение false
    inputs: KB, база знаний — множество определенных выражений
              первого порядка
         $\alpha$ , запрос — атомарное высказывание
    local variables: new, новые высказывания, выводимые
                      в каждой итерации
    repeat until множество new не пусто
        new  $\leftarrow \{\}$ 
        for each высказывание r in KB do
             $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{Standardize-Apart}(r)$ 
            for each подстановка  $\theta$ , такая что  $\text{Subst}(\theta, p_1 \wedge \dots \wedge p_n) =$ 
                 $\text{Subst}(\theta, p'_1 \wedge \dots \wedge p'_n)$  для некоторых  $p'_1, \dots, p'_n$ 
                в базе знаний KB
             $q' \leftarrow \text{Subst}(\theta, q)$ 
            if выражение  $q'$  не является переименованием
                некоторого высказывания, которое уже находится
                в KB, или рассматривается как элемент множества
            new then do
                добавить  $q'$  к множеству new
             $\phi \leftarrow \text{Unify}(q', \alpha)$ 
            if значение  $\phi$  не представляет собой fail
                then return  $\phi$ 
            добавить множество new к базе знаний KB
    return false

```

---

Для иллюстрации работы алгоритма FOL-FC-Ask воспользуемся описанной выше задачей доказательства преступления. Импликационными высказываниями являются высказывания, приведенные в уравнениях 9.3, 9.6–9.8. Требуются следующие две итерации.

- В первой итерации правило 9.3 имеет невыполненные предпосылки.

Правило 9.6 выполняется с подстановкой  $\{x/M_1\}$  и добавляется высказывание  $Sells(West, M_1, Nono)$ .

Правило 9.7 выполняется с подстановкой  $\{x/M_1\}$  и добавляется высказывание  $Weapon(M_1)$ .

Правило 9.8 выполняется с подстановкой  $\{x/Nono\}$  и добавляется высказывание  $Hostile(Nono)$ .

- На второй итерации правило 9.3 выполняется с подстановкой  $\{x/West, y/M_1, z/Nono\}$  и добавляется высказывание  $Criminal(West)$ .

Сформированное дерево доказательства показано на рис. 9.2. Обратите внимание на то, что в этот момент невозможны какие-либо новые логические выводы, поскольку каждое высказывание, заключение которого можно было бы найти с помощью прямого логического вывода, уже явно содержится в базе знаний KB. Такое состояние базы знаний называется **фиксированной точкой** (fixed point) в процессе логического вывода. Фиксированные точки, достигаемые при прямом логическом выводе с использованием определенных выражений первого порядка, аналогичны фиксированным точкам, возникающим при пропозициональном прямом логическом выводе (с. 315); основное различие состоит в том, что фиксированная точка первого порядка может включать атомарные высказывания с квантором всеобщности.

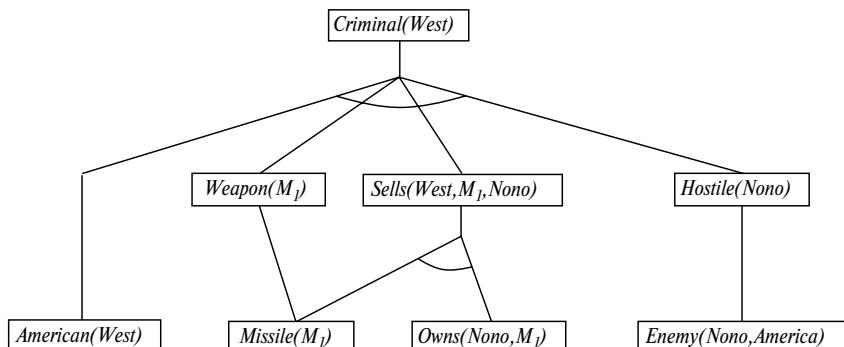


Рис. 9.2. Дерево доказательства, сформированное путем прямого логического вывода в примере доказательства преступления. Первоначальные факты показаны на нижнем уровне, факты, выведенные логическим путем в первой итерации, — на среднем уровне, а факт, логически выведенный во второй итерации, — на верхнем уровне

Свойства алгоритма FOL-FC-Ask проанализировать несложно. Во-первых, он является **непротиворечивым**, поскольку каждый этап логического вывода представляет собой применение обобщенного правила отделения, которое само является непротиворечивым. Во-вторых, он является **полным** применительно к базам знаний

с определенными выражениями; это означает, что он позволяет ответить на любой запрос, ответы на который следуют из базы знаний с определенными выражениями. Для баз знаний Datalog, которые не содержат функциональных символов, доказательство полноты является довольно простым. Начнем с подсчета количества возможных фактов, которые могут быть добавлены, определяющего максимальное количество итераций. Допустим, что  $k$  — максимальная **арность** (количество параметров) любого предиката,  $p$  — количество предикатов и  $n$  — количество константных символов. Очевидно, что может быть не больше чем  $pn^k$  различных базовых фактов, поэтому алгоритм должен достичь фиксированной точки именно после стольких итераций. В таком случае можно применить обоснование приведенного выше утверждения, весьма аналогичное доказательству полноты пропозиционального прямого логического вывода (см. с. 315). Подробные сведения о том, как осуществить переход от пропозициональной полноты к полноте первого порядка, приведены применительно к алгоритму резолюции в разделе 9.5.

При его использовании к более общим определенным выражениям с функциональными символами алгоритм FOL-FC-Ask может вырабатывать бесконечно большое количество новых фактов, поэтому необходимо соблюдать исключительную осторожность. Для того случая, в котором из базы знаний следует ответ на высказывание запроса  $q$ , необходимо прибегать к использованию теоремы Эрбрана для обеспечения того, чтобы алгоритм мог найти доказательство (случай, касающийся резолюции, описан в разделе 9.5). А если запрос не имеет ответа, то в некоторых случаях может оказаться, что не удается нормально завершить работу данного алгоритма. Например, если база знаний включает аксиомы Пеано:

$$\begin{aligned} &\text{NatNum}(0) \\ &\forall n \text{ NatNum}(n) \Rightarrow \text{NatNum}(S(n)) \end{aligned}$$

то в результате прямого логического вывода будут добавлены факты  $\text{NatNum}(S(0))$ ,  $\text{NatNum}(S(S(0)))$ ,  $\text{NatNum}(S(S(S(0))))$  и т.д. Вообще говоря, избежать возникновения этой проблемы невозможно. Как и в общем случае логики первого порядка, задача определения того, следуют ли высказывания из базы знаний, сформированной с использованием определенных выражений, является полуразрешимой.

### Эффективный прямой логический вывод

Алгоритм прямого логического вывода, приведенный в листинге 9.2, был спроектирован не с целью обеспечения эффективного функционирования, а, скорее, с целью упрощения его понимания. Существуют три возможных источника осложнений в его работе. Во-первых, “внутренний цикл” этого алгоритма предусматривает поиск всех возможных унификаторов, таких, что предпосылка некоторого правила унифицируется с подходящим множеством фактов в базе знаний. Такая операция часто именуется **согласованием с шаблоном** и может оказаться очень дорогостоящей. Во-вторых, в этом алгоритме происходит повторная проверка каждого правила в каждой итерации для определения того, выполняются ли его предпосылки, даже если в базу знаний в каждой итерации вносится лишь очень немного дополнений. В-третьих, этот алгоритм может вырабатывать много фактов, которые не имеют отношения к текущей цели. Устраним каждый из этих источников неэффективности по очереди.

### Согласование правил с известными фактами

Проблема согласования предпосылки правила с фактами, хранящимися в базе знаний, может показаться достаточно простой. Например, предположим, что требуется применить следующее правило:

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

Для этого необходимо найти все факты, которые согласуются с выражением  $\text{Missile}(x)$ ; в базе знаний, индексированной подходящим образом, это можно выполнить за постоянное время в расчете на каждый факт. А теперь рассмотрим правило, подобное следующему:

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$$

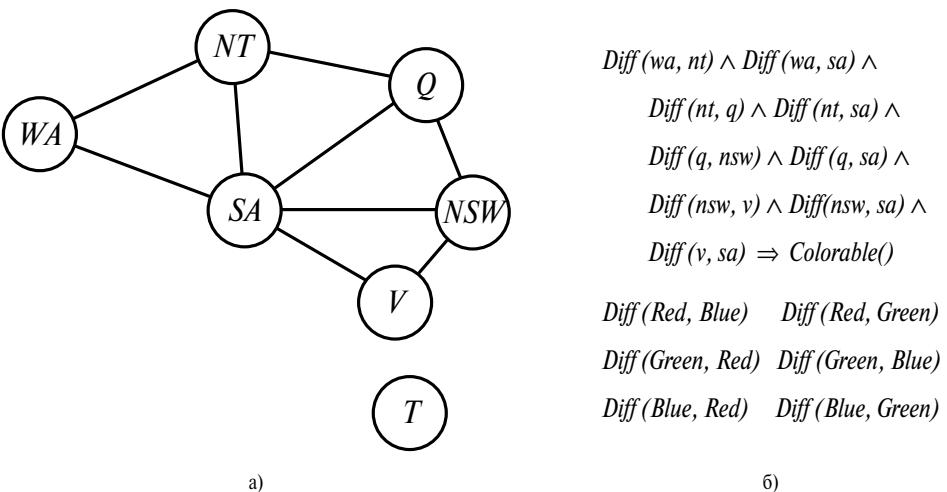
Найти все объекты, принадлежащие государству Ноуно, опять-таки можно за постоянное время в расчете на каждый объект; затем мы можем применить к каждому объекту проверку, является ли он ракетой. Но если в базе знаний содержится много сведений об объектах, принадлежащих государству Ноуно, и лишь немного данных о ракетах, то было бы лучше вначале найти все ракеты, а затем проверить, какие из них принадлежат Ноуно. Это — проблема ~~с~~ **упорядочения конъюнктов**: поиск упорядочения, позволяющего решать конъюнкты в предпосылке правила таким образом, чтобы общая стоимость решения была минимальной. Как оказалось, задача поиска оптимального упорядочения является NP-трудной, но имеются хорошие эвристики. Например, эвристика **с наибольшей ограниченной переменной**, применявшаяся при решении задач CSP в главе 5, подсказывает, что необходимо упорядочить конъюнкты так, чтобы вначале проводился поиск ракет, если количество ракет меньше по сравнению с количеством всех известных объектов, принадлежащих государству Ноуно.

Между процедурами согласования с шаблоном и удовлетворения ограничений действительно существует очень тесная связь. Каждый конъюнкт может рассматриваться как ограничение на содержащиеся в нем переменные; например,  $\text{Missile}(x)$  — это унарное ограничение на  $x$ . Развивая эту идею, можно прийти к выводу, что ~~с~~ **существует возможность представить любую задачу CSP с конечной областью определения как единственное определенное выражение наряду с некоторыми ка-сающимися ее базовыми фактами**. Рассмотрим приведенную на рис. 5.1 задачу раскраски карты, которая снова показана на рис. 9.3, а. Эквивалентная формулировка в виде одного определенного выражения приведена на рис. 9.3, б. Очевидно, что заключение  $\text{Colorable}()$  можно вывести из этой базы знаний, только если данная задача CSP имеет решение. А поскольку задачи CSP, вообще говоря, включают задачи 3-SAT в качестве частных случаев, на основании этого можно сделать вывод, что ~~с~~ **задача согла-сования определенного выражения с множеством фактов является NP-трудной**.

То, что во внутреннем цикле алгоритма прямого логического вывода приходится решать NP-трудную задачу согласования, может показаться на первый взгляд довольно неприятным. Тем не менее, есть следующие три фактора, благодаря которым эта проблема предстает немного в лучшем свете.

- Напомним, что большинство правил в базах знаний, применяемых на практике, являются небольшими и простыми (подобно правилам, используемым в примере доказательства преступления), а не большими и сложными (как в формулировке задачи CSP, приведенной на рис. 9.3). В мире пользователей

баз данных принято считать, что размеры правил и арности предикатов не превышают некоторого постоянного значения, и принимать во внимание только ~~сложность~~ сложность данных, т.е. сложность логического вывода как функции от количества базовых фактов в базе данных. Можно легко показать, что обусловленная данными сложность в прямом логическом выводе определяется полиномиальной зависимостью.



*Рис. 9.3. Иллюстрация связи между процессами согласования с шаблоном и удовлетворения ограничений: граф ограничений для раскрашивания карты Австралии (см. рис. 5.1) (а); задача CSP раскрашивания карты, представленная в виде единственного определенного выражения (б). Обратите внимание на то, что области определения переменных заданы неявно с помощью констант, приведенных в базовых фактах для предиката Diff*

- Могут рассматриваться подклассы правил, для которых согласование является наиболее эффективным. По сути, каждое выражение на языке Datalog может рассматриваться как определяющее некоторую задачу CSP, поэтому согласование будет осуществимым только тогда, когда соответствующая задача CSP является разрешимой. Некоторые разрешимые семейства задач CSP описаны в главе 5. Например, если граф ограничений (граф, узлами которого являются переменные, а дугами — ограничения) образует дерево, то задача CSP может быть решена за линейное время. Точно такой же результат остается в силе для согласования с правилами. Например, если из карты, приведенной на рис. 9.3, будет удален узел *SA*, относящийся к Южной Австралии, то результирующее выражение примет следующий вид:

$$Diff(wa, nt) \wedge Diff(nt, q) \wedge Diff(q, nsw) \wedge Diff(nsw, v) \Rightarrow Colorable()$$

что соответствует сокращенной задаче CSP, показанной на рис. 5.7. Для решения задачи согласования с правилами могут непосредственно применяться алгоритмы решения задач CSP с древовидной структурой.

- Можно приложить определенные усилия по устраниению излишних попыток согласования с правилами в алгоритме прямого логического вывода, что является темой следующего раздела.

### Инкрементный прямой логический вывод

Когда авторы демонстрировали в предыдущем разделе на примере доказательства преступления, как действует прямой логический вывод, они немного схитрили; в частности, не показали некоторые из согласований с правилами, выполняемые алгоритмом, приведенным в листинге 9.2. Например, во второй итерации правило

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

согласуется с фактом  $\text{Missile}(M_1)$  (еще раз), и, безусловно, при этом ничего не происходит, поскольку заключение  $\text{Weapon}(M_1)$  уже известно. Таких излишних согласований с правилами можно избежать, сделав следующее наблюдение: *каждый новый факт, выведенный в итерации  $t$ , должен быть получен по меньшей мере из одного нового факта, выведенного в итерации  $t-1$* . Это наблюдение соответствует истине, поскольку любой логический вывод, который не требовал нового факта из итерации  $t-1$ , уже мог быть выполнен в итерации  $t-1$ .

Такое наблюдение приводит естественным образом к созданию алгоритма инкрементного прямого логического вывода, в котором в итерации  $t$  проверка правила происходит, только если его предпосылка включает конъюнкт  $p_i$ , который унифицируется с фактом  $p_i'$ , вновь выведенным в итерации  $t-1$ . Затем на этапе согласования с правилом значение  $p_i$  фиксируется для согласования с  $p_i'$ , но при этом допускается, чтобы остальные конъюнкты в правиле согласовывались с фактами из любой предыдущей итерации. Этот алгоритм в каждой итерации вырабатывает точно такие же факты, как и алгоритм, приведенный в листинге 9.2, но является гораздо более эффективным.

При использовании подходящей индексации можно легко выявить все правила, которые могут быть активизированы любым конкретным фактом. И действительно, многие реальные системы действуют в режиме “обновления”, при котором прямой логический вывод происходит в ответ на каждый новый факт, сообщенный системе с помощью операции `Tell`. Операции логического вывода каскадно распространяются через множество правил до тех пор, пока не достигается фиксированная точка, а затем процесс начинается снова, вслед за поступлением каждого нового факта.

Как правило, в результате добавления каждого конкретного факта в действительности активизируется лишь небольшая доля правил в базе знаний. Это означает, что при повторном конструировании частичных согласований с правилами, имеющими некоторые невыполненные предпосылки, выполняется существенный объем ненужной работы. Рассматриваемый здесь пример доказательства преступления слишком мал, чтобы на нем можно было наглядно показать такую ситуацию, но следует отметить, что частичное согласование конструируется в первой итерации между следующим правилом:

$$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$$

и фактом  $\text{American}(\text{West})$ . Затем это частичное согласование отбрасывается и снова формируется во второй итерации (в которой данное правило согласуется успешно). Было бы лучше сохранять и постепенно дополнять частичные согласования по мере поступления новых фактов, а не отбрасывать их.

В **rete-алгоритме**<sup>3</sup> была впервые предпринята серьезная попытка решить эту проблему. Алгоритм предусматривает предварительную обработку множества пра-

<sup>3</sup> Здесь *rete* — латинское слово, которое переводится как сеть и читается по-русски “рете”, а по-английски — “рити”.

вил в базе знаний для формирования своего рода сети потока данных (называемой *rete*-сетью), в которой каждый узел представляет собой литерал из предпосылки какого-либо правила. По этой сети распространяются операции связывания переменных и останавливаются после того, как в них не удается выполнить согласование с каким-то литералом. Если в двух литералах некоторого правила совместно используется какая-то переменная (например,  $Sells(x, y, z) \wedge Hostile(z)$  в примере доказательства преступления), то варианты связывания из каждого литерала пропускаются через узел проверки равенства. Процессу связывания переменных, достигших узла *n*-арного литерала, такого как  $Sells(x, y, z)$ , может потребоваться перейти в состояние ожидания того, что будут определены связывания для других переменных, прежде чем он сможет продолжаться. В любой конкретный момент времени состояние *rete*-сети охватывает все частичные согласования с правилами, что позволяет избежать большого объема повторных вычислений.

Не только сами *rete*-сети, но и различные их усовершенствования стали ключевым компонентом так называемых **продукционных систем**, которые принадлежат к числу самых первых систем прямого логического вывода, получивших широкое распространение<sup>4</sup>. В частности, с использованием архитектуры производственной системы была создана система Xcon (которая первоначально называлась R1) [1026]. Система Xcon содержала несколько тысяч правил и предназначалась для проектирования конфигураций компьютерных компонентов для заказчиков Digital Equipment Corporation. Ее создание было одним из первых очевидных успешных коммерческих проектов в развивающейся области экспертных систем. На основе той же базовой технологии, которая была реализована на языке общего назначения Ops-5, было также создано много других подобных систем.

Кроме того, производственные системы широко применяются в **когнитивных архитектурах** (т.е. моделях человеческого мышления), в таких как ACT [31] и Soar [880]. В подобных системах “рабочая память” системы моделирует кратковременную память человека, а продукция образуют часть долговременной памяти. В каждом цикле функционирования происходит согласование продукции с фактами из рабочей памяти. Продукции, условия которых выполнены, могут добавлять или удалять факты в рабочей памяти. В отличие от типичных ситуаций с большим объемом данных, наблюдаемых в базах данных, производственные системы часто содержат много правил и относительно немного фактов. При использовании технологии согласования, оптимизированной должным образом, некоторые современные системы могут оперировать в реальном времени больше чем с миллионом правил.

### Не относящиеся к делу факты

Последний источник неэффективности прямого логического вывода, по-видимому, свойствен самому этому подходу и также возникает в контексте пропозициональной логики (см. раздел 7.5). Прямой логический вывод предусматривает выполнение всех допустимых этапов логического вывода на основе всех известных фактов, даже если они не относятся к рассматриваемой цели. В примере доказательства преступления не было правил, способных приводить к заключениям, не относящимся к делу, поэтому отсутствие направленности не вызывало каких-либо проблем. В других случаях (например, если бы в базу знаний было внесено несколь-

<sup>4</sup> Слово **продукция** в названии **производственная система** обозначает правило “условие-действие”.

ко правил с описанием кулинарных предпочтений американцев и цен на ракеты) алгоритм FOL-FC-Ask вырабатывал бы много нерелевантных заключений.

Один из способов предотвращения формирования нерелевантных заключений состоит в использовании обратного логического вывода, как описано в разделе 9.4. Еще одно, второе решение состоит в том, чтобы ограничить прямой логический вывод избранным подмножеством правил; этот подход обсуждался в контексте пропозициональной логики. Третий подход сформировался в сообществе пользователей дедуктивных баз данных, для которых прямой логический вывод является стандартным инструментальным средством. Идея этого подхода состоит в том, чтобы перезаписывать множество правил с использованием информации из цели так, что в процессе прямого логического вывода рассматриваются только релевантные связывания переменных (принадлежащие к так называемому **магическому множеству**). Например, если целью является *Criminal(West)*, то правило, приводящее к заключению *Criminal(x)*, может быть перезаписано для включения дополнительного конъюнкта, ограничивающего значение *x*, следующим образом:

$$\begin{aligned} Magic(x) \wedge American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge \\ Hostile(z) \Rightarrow Criminal(x) \end{aligned}$$

Факт *Magic(West)* также добавляется в базу знаний. Благодаря этому в процессе прямого логического вывода будет рассматриваться только факт о полковнике Уэсте, даже если база знаний содержит факты о миллионах американцев. Полный процесс определения магических множеств и перезаписи базы знаний является слишком сложным для того, чтобы мы могли заняться в этой главе его описанием, но основная идея состоит в том, что выполняется своего рода “универсальный” обратный логический вывод от цели для выяснения того, какие связывания переменных нужно будет ограничивать. Поэтому подход с использованием магических множеств может рассматриваться как гибридный между прямым логическим выводом и обратной предварительной обработкой.

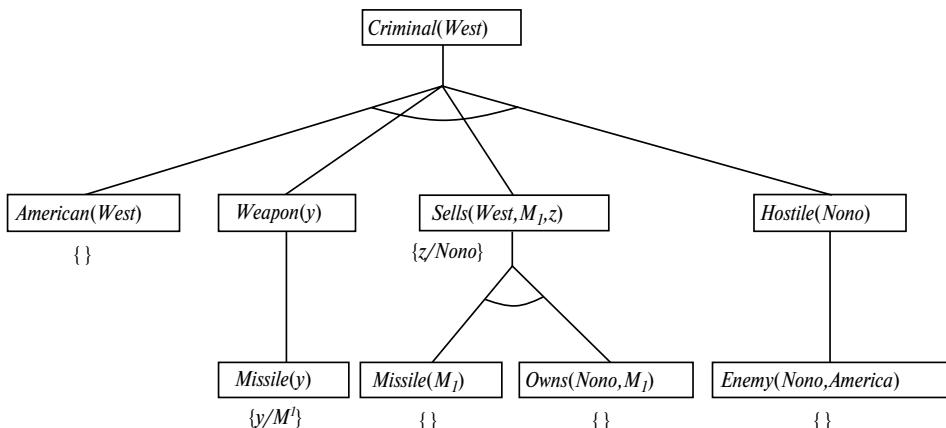
## 9.4. ОБРАТНЫЙ ЛОГИЧЕСКИЙ ВЫВОД

Во втором большом семействе алгоритмов логического вывода используется подход с **обратным логическим выводом**, представленный в разделе 7.5. Эти алгоритмы действуют в обратном направлении, от цели, проходя по цепочке от одного правила к другому, чтобы найти известные факты, которые поддерживают доказательство. Мы опишем основной алгоритм, а затем покажем, как он используется в **логическом программировании**, представляющем собой наиболее широко применяемую форму автоматизированного формирования рассуждений. В этом разделе будет также показано, что обратный логический вывод имеет некоторые недостатки по сравнению с прямым логическим выводом, и описаны некоторые способы преодоления этих недостатков. Наконец, будет продемонстрирована тесная связь между логическим программированием и задачами удовлетворения ограничений.

### Алгоритм обратного логического вывода

Простой алгоритм обратного логического вывода, FOL-BC-Ask, приведен в листинге 9.3. Он вызывается со списком целей, содержащим единственный элемент

(первоначальный запрос) и возвращает множество всех подстановок, которые удовлетворяют этому запросу. Список целей можно рассматривать как “стек” целей, ожидающих отработки; если все они могут быть выполнены, то текущая ветвь доказательства формируется успешно. В алгоритме берется первая цель из списка и выполняется поиск в базе знаний всех выражений, положительный литерал которых (или **голова**) унифицируется с целью. При обработке каждого такого выражения создается новый рекурсивный вызов, в котором предпосылки (или **тело**) выражения добавляются к стеку целей. Напомним, что факты представляют собой выражения с головой, но без тела, поэтому, если какая-то цель унифицируется с известным фактом, то к стеку не добавляются какие-то подцели, а сама эта цель считается получившей решение. На рис. 9.4 показано дерево доказательства для получения факта *Criminal(West)* из высказываний, приведенных в уравнениях 9.3–9.10.



*Рис. 9.4. Дерево доказательства, сформированное путем обратного логического вывода для доказательства того, что полковник Уэст совершил преступление. Это дерево следует читать в глубину, слева направо. Чтобы доказать факт *Criminal(West)*, необходимо доказать четыре конъюнкта, находящихся под ним. Некоторые из них находятся в базе знаний, а другие требуют дальнейшего обратного логического вывода. Связывания для каждой успешной унификации показаны после соответствующей подцели. Обратите внимание на, что после успешного достижения одной подцели в конъюнкции ее подстановка применяется для последующих подцелей. Таким образом, к тому времени, как алгоритм FOL-BC-Ask достигает последнего конъюнкта, первоначально имевшего форму *Hostile(z)*, переменная *z* уже связана с *Nono*.*

### Листинг 9.3. Простой алгоритм обратного логического вывода

```

function FOL-BC-Ask(KB, goals, θ) returns множество подстановок
  inputs: KB, база знаний
          goals, список конъюнктов, образующих запрос (подстановка θ уже применена)
          θ, текущая подстановка, первоначально пустая подстановка {}
  local variables: answers, ответы – множество подстановок,
          первоначально пустое

  if список goals пуст then return {θ}
  q' ← Subst(θ, First(goals))

```

---

```

for each высказывание  $r$  in KB, где Standardize-Apart( $r$ ) =
    ( $p_1 \wedge \dots \wedge p_n \Rightarrow q$ ) и  $\theta' \leftarrow \text{Unify}(q, q')$  является
    выполнимым
    new_goals  $\leftarrow [p_1, \dots, p_n | \text{Rest(goals)}]$ 
    answers  $\leftarrow \text{FOL-BC-Ask}(KB, \text{new\_goals}, \text{Compose}(\theta', \theta)) \cup \text{answers}$ 
return answers

```

---

В этом алгоритме используется **композиция** подстановок. Здесь  $\text{Compose}(\theta_1, \theta_2)$  — это подстановка, результат которой идентичен результату применения каждой подстановки по очереди, следующим образом:

$$\text{Subst}(\text{Compose}(\theta_1, \theta_2), p) = \text{Subst}(\theta_2, \text{Subst}(\theta_1, p))$$

В данном алгоритме текущие связывания переменных, которые хранятся в подстановке  $\theta$ , компонуются со связываниями, возникающими в результате унификации цели с головой выражения, что приводит к получению нового множества текущих связываний для рекурсивного вызова.

Алгоритм обратного логического вывода в том виде, в каком он был приведен в этом разделе, безусловно, представляет собой алгоритм поиска в глубину. Это означает, что его потребности в пространстве линейно зависят от размера доказательства (если на данный момент пренебречь тем, какой объем пространства требуется для накопления решений). Это также означает, что обратный логический вывод (в отличие от прямого логического вывода) страдает от проблем, обусловленных наличием повторяющихся состояний и неполноты. Эти проблемы и некоторые потенциальные решения будут рассматриваться ниже, но вначале покажем, как обратный логический вывод используется в системах логического программирования.

## Логическое программирование

Логическое программирование — это технология, позволяющая довольно близко приблизиться к воплощению декларативного идеала, описанного в главе 7, согласно которому системы должны конструироваться путем представления знаний на некотором формальном языке, а задачи решаться путем применения процессов логического вывода к этим знаниям. Такой идеал выражен в следующем уравнении Роберта Ковальского:

$$\text{Алгоритм} = \text{Логика} + \text{Управление}$$

Одним из языков логического программирования, намного превосходящим все прочие по своей распространенности, является **Prolog**. Количество его пользователей насчитывает сотни тысяч. Он используется в основном в качестве языка быстрой разработки прототипов, а также служит для решения задач символьических манипуляций, таких как написание компиляторов [1536] и синтаксический анализ текстов на естественном языке [1208]. На языке Prolog было написано много экспертных систем для юридических, медицинских, финансовых и других проблемных областей.

Программы Prolog представляют собой множества определенных выражений, записанных в системе обозначений, немного отличающейся от используемой стандартной логики первого порядка. В языке Prolog прописные буквы применяются для обозначения переменных, а строчные — для обозначения констант. Выражения записываются с головой, предшествующей телу; символ `:` — служит для обозначения

импликации, направленной влево, запятые разделяют литералы в теле, а точка обозначает конец высказывания, как показано ниже.

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

Язык Prolog включает “синтаксические упрощения” (syntactic sugar) для обозначения списков и арифметических выражений. Например, ниже приведена программа Prolog для предиката `append(X, Y, Z)`, которая выполняется успешно, если список `Z` представляет собой результат дополнения списка `Y` списком `X`.

```
append([], Y, Y).
append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

На естественном языке эти выражения можно прочитать так: во-первых, дополнение списка `Y` пустым списком приводит к получению того же списка `Y`, и, во-вторых, `[A|Z]` — это результат дополнения списка `Y` списком `[A|X]`, при условии, что `Z` — это результат дополнения списка `Y` списком `X`. Такое определение предиката `append` на первый взгляд кажется весьма подобным соответствующему определению на языке Lisp, но фактически является гораздо более мощным. Например, в систему можно ввести запрос `append(A, B, [1, 2])` — какие два списка можно дополнить один другим, чтобы получить `[1, 2]`? Система возвратит следующие решения:

```
A=[]      B=[1,2]
A=[1]     B=[2]
A=[1,2]   B=[]
```

Выполнение программ Prolog осуществляется по принципу обратного логического вывода с поиском в глубину, при котором попытка применения выражений выполняется в том порядке, в каком они записаны в базу знаний. Но некоторые описанные ниже особенности языка Prolog выходят за рамки стандартного логического вывода.

- В нем предусмотрено множество встроенных функций для выполнения арифметических операций. Литералы, в которых используются соответствующие функциональные символы, “доказываются” путем выполнения кода, а не осуществления дальнейшего логического вывода. Например, цель “`X is 4+3`” достигается успешно после связывания переменной `X` со значением 7. С другой стороны, попытка достижения цели “`5 is X+Y`” оканчивается неудачей, поскольку эти встроенные функции не обеспечивают решения произвольных уравнений<sup>5</sup>.
- В языке предусмотрены встроенные предикаты, вызывающие при их выполнении побочные эффекты. К ним относятся предикаты ввода-вывода и предикаты `assert/retract` для модификации базы знаний. Такие предикаты не имеют аналогов в логике и могут порождать некоторые эффекты, вызывающие путаницу, например, если факты подтверждаются (и вводятся в базу знаний) некоторой ветвью дерева доказательства, которая в конечном итоге оканчивается неудачей.

---

<sup>5</sup> Следует отметить, что если бы в некоторой программе Prolog были предусмотрены аксиомы Пеано, то такие цели могли быть решены с помощью логического вывода.

- В языке Prolog допускается определенная форма отрицания —**отрицание как недостижение цели**. Отрицаемая цель `not P` считается доказанной, если системе не удается доказать `P`. Таким образом, следующее высказывание:  
`alive(X) :- not dead(X).`  
можно прочитать так: “Любого следует считать живым, если нельзя доказать, что он мертв”.
- В языке Prolog предусмотрен оператор равенства, “`=`”, но он не обладает всей мощью логического равенства. Цель с оператором равенства достигается успешно, если в ней два терма являются унифицируемыми, а в противном случае попытка ее достижения оканчивается неудачей. Таким образом, цель `X+Y=2+3` достигается успешно, после связывания переменной `X` со значением 2, а `Y` — со значением 3, а попытка достижения цели `morningstar=eveningstar` оканчивается неудачей. (В классической логике последнее равенство может быть или не быть истинным.) Не могут быть подтверждены (введены в базу знаний) какие-либо факты или правила, касающиеся равенства.
- Из алгоритма унификации Prolog исключена **проверка вхождения**. Это означает, что могут быть сделаны некоторые противоречивые логические выводы; такая проблема возникает редко, за исключением тех ситуаций, когда язык Prolog используется для доказательства математических теорем.

Решения, принятые при проектировании языка Prolog, представляют собой компромисс между стремлениями обеспечить декларативность и вычислительную эффективность (по крайней мере, эффективность в той ее трактовке, которая существовала в период разработки языка Prolog). Мы вернемся к этой теме после рассмотрения того, как реализована система Prolog.

## Эффективная реализация логических программ

Подготовка программ Prolog к выполнению может осуществляться в двух режимах: интерпретация и компиляция. Интерпретация по существу представляет собой применение алгоритма FOL-BC-Ask, приведенного в листинге 9.3, к программе, представленной в виде базы знаний. Предыдущее предложение включает слова “по существу”, поскольку в интерпретаторах Prolog предусмотрены всевозможные усовершенствования, предназначенные для максимального повышения скорости работы. Здесь рассматриваются только два таких усовершенствования.

Во-первых, вместо формирования списка всех возможных ответов для каждой подцели перед переходом к следующей подцели интерпретаторы Prolog вырабатывают один ответ и “дают обещание” выработать остальные после того, как будет полностью исследован текущий ответ. Такое “обещание” оформляется как **точка выбора** (`choice point`). После того как в процессе поиска в глубину завершается исследование возможных решений, вытекающих из текущего ответа, и происходит возврат к точке выбора, в этой точке используемые структуры данных дополняются, чтобы в них можно было включить новый ответ для данной подцели и сформировать новую точку выбора. Такой подход позволяет экономить и время, и пространство, а также обеспечивает создание очень простого интерфейса для отладки, поскольку постоянно существует лишь единственный путь решения, подлежащий рассмотрению.

Во-вторых, в приведенной в данной главе простой реализации алгоритма FOL-BC-Ask много времени затрачивается на выработку и компоновку подстановок. В языке Prolog подстановки реализуются с использованием логических переменных, позволяющих сохранить в памяти их текущее связывание. В любой момент времени каждая переменная в программе является либо несвязанной, либо связанной с некоторым значением. Эти переменные и значения, вместе взятые, неявно определяют подстановку для текущей ветви доказательства. Любая попытка продления пути позволяет лишь добавить новые связывания переменных, поскольку стремление ввести другое связывание для уже связанной переменной приводит к неудачному завершению унификации. После того как попытка продления пути поиска оканчивается неудачей, система Prolog возвращается к предыдущей точке выбора и только после этого получает возможность отменить связывания некоторых переменных. Такая операция отмены выполняется благодаря тому, что данные обо всех переменных, для которых было выполнено связывание, отслеживаются в стеке, называемом **контрольным стеком** (trail). По мере того как в функции Unify-Var осуществляется связывание каждой новой переменной, эта переменная задвигается в контрольный стек. Если попытка достижения некоторой цели оканчивается неудачей и наступает время возвратиться к предыдущей точке пункта выбора, отменяется связывание каждой из этих переменных по мере их выталкивания из контрольного стека.

Но даже в самых эффективных интерпретаторах Prolog, в связи с издержками на поиск по индексу, унификацию и формирование стека рекурсивных вызовов, требуется выполнение нескольких тысяч машинных команд в расчете на каждый этап логического вывода. В действительности интерпретатор постоянно ведет себя так, как если бы он никогда до сих пор не видел данную программу; например, ему приходится каждый раз находить выражения, которые согласуются с целью. С другой стороны, откомпилированная программа Prolog представляет собой процедуру логического вывода для конкретного множества выражений, поэтому ей известно, какие выражения согласуются с целью. В процессе компиляции система Prolog по сути формирует миниатюрную программу автоматического доказательства теоремы для каждого отдельного предиката, устранив тем самым основную часть издержек интерпретации. Эта система позволяет также применять **открытый код** для процедуры унификации каждого отдельного вызова, что позволяет избежать необходимости проведения явного анализа структуры терма (подробные сведения об унификации с открытым кодом приведены в [1557]).

Наборы команд современных компьютеров плохо согласуются с семантикой Prolog, поэтому большинство компиляторов Prolog компилирует программу в промежуточный язык, а не непосредственно в машинный язык. Наиболее широко применяемым промежуточным языком является язык WAM (Warren Abstract Machine — абстрактная машина Уоррена) получивший название в честь Дэвида Г.Д. Уоррена, одного из создателей первого компилятора Prolog. Язык WAM представляет собой абстрактное множество команд, которое подходит для преобразования в него программы Prolog и может интерпретироваться или транслироваться в машинный язык. Другие компиляторы транслируют программу Prolog в программу на языке высокого уровня, таком как Lisp или C, а затем используют компилятор этого языка для трансляции в машинный язык. Например, определение предиката Append может быть откомпилировано в код, показанный в листинге 9.4. Ниже приведено несколько замечаний, заслуживающих упоминания в этой связи.

**Листинг 9.4.** Псевдокод, представляющий собой результат компиляции предиката `Append`. Функция `New-Variable` возвращает новую переменную, отличную от всех других переменных, использовавшихся до сих пор. Процедура `Call(continuation)` продолжает выполнение с заданным продолжением `continuation`

---

```

procedure Append(ax, y, az, continuation)

    trail ← Global-Trail-Pointer()
    if ax = [] and Unify(y, az) then Call(continuation)
    Reset-Trail(trail)
    a ← New-Variable(); x ← New-Variable(); z ← New-Variable()
    if Unify(ax, [a | x]) and Unify(az, [a | z])
        then Append(x, y, z, continuation)

```

---

- Выражения предиката `Append` преобразуются в процедуру, а этапы логического вывода осуществляются путем вызова этой процедуры, поэтому не приходится выполнять поиск соответствующих выражений в базе знаний.
- Как было описано выше, текущие связывания переменных хранятся в контролльном стеке. На первом этапе выполнения этой процедуры текущее состояние контролльного стека сохраняется в памяти, поэтому оно может быть восстановлено с помощью функции `Reset-Trail`, если попытка выполнения первого выражения окончится неудачей. Это приводит к отмене всех связываний, сформированных при первом вызове процедуры `Unify`.
- Сложнейшей частью этой программы является использование **продолжений** для реализации точек выбора. *Продолжение* может рассматриваться как структура данных, в которой упакованы процедура и список параметров, вместе взятые, определяющая, что следует делать дальше, после успешного достижения текущей цели. Дело в том, что после достижения цели не было бы достаточно просто возвратить управление из процедуры, подобной `Append`, поскольку успех может быть достигнут несколькими способами, и каждый из них должен быть исследован. Параметр `continuation`, определяющий продолжение, позволяет решить эту проблему, поскольку он может быть вызван после каждого успешного достижения цели. В приведенном здесь коде процедуры `Append`, если первый параметр является пустым, то предикат `Append` достиг успеха. Затем вызывается продолжение с помощью процедуры `Call` (притом что в контролльном стеке находятся все подходящие связывания) для того, чтобы определить, что делать дальше. Например, если процедура `Append` вызвана на верхнем уровне дерева доказательства, то структура данных `continuation` будет использоваться для вывода информации о связывании переменных.

До того как Уоррен выполнил свою работу по внедрению компиляции в системы логического вывода на языке Prolog, средства логического программирования работали слишком медленно для того, чтобы они действительно могли найти широкое применение. Компиляторы, разработанные Уорреном и другими специалистами, позволили достичь скоростей обработки кода Prolog, способных конкурировать с языком С в самых различных стандартных эталонных тестах [1536]. Безусловно, тот факт, что теперь появилась возможность написать планировщик или синтаксиче-

ский анализатор текста на естественном языке, состоящий из нескольких десятков строк на языке Prolog, говорит о том, что этот язык становится более подходящим (по сравнению с языком С) для создания прототипов большинства исследовательских проектов в области искусственного интеллекта небольших масштабов.

Значительное повышение скорости позволяет также обеспечить распараллеливание работы. Организация параллельного выполнения может осуществляться по двум направлениям. Первое направление, получившее название **ИЛИ-параллелизма**, исходит из возможности унификации цели со многими различными выражениями в базе знаний. Каждая из этих операций унификации приводит к появлению независимой ветви в пространстве поиска, которая может привести к потенциальному решению, и поиск решений по всем таким ветвям может осуществляться параллельно. Второе направление, называемое **И-параллелизмом**, исходит из возможности параллельного решения каждого конъюнкта в теле некоторой импликации. Задача достижения И-параллелизма является более сложной, поскольку для поиска решений всей конъюнкции требуются совместимые связывания для всех переменных. Поэтому при обработке каждой конъюнктивной ветви необходимо обеспечивать обмен данными с другими ветвями для гарантированного достижения глобального решения.

### Избыточный логический вывод и бесконечные циклы

Теперь рассмотрим, в чем состоит ахиллесова пята языка Prolog: несогласованность между организацией поиска в глубину и деревьями поиска, которые включают повторяющиеся состояния и бесконечные пути. Рассмотрим следующую логическую программу, позволяющую определить, существует ли путь между двумя точками в ориентированном графе:

```
path(X, Z) :- link(X, Z).
path(X, Z) :- path(X, Y), link(Y, Z).
```

На рис. 9.5, *a* приведен простой граф, состоящий из трех узлов, который описан с помощью фактов `link(a, b)` и `link(b, c)`. При использовании этой программы запрос `path(a, c)` вырабатывает дерево доказательства, показанное на рис. 9.6, *a*. С другой стороны, если эти два выражения расположены в таком порядке:

```
path(X, Z) :- path(X, Y), link(Y, Z).
path(X, Z) :- link(X, Z).
```

то система Prolog следует по бесконечному пути, как показано на рис. 9.6, *b*. Поэтому система Prolog является **неполной**, как и программа автоматического доказательства теоремы для определенных выражений (как показано в этом примере, даже применительно к программам, соответствующим формату языка Datalog), поскольку для некоторых баз знаний эта система оказывается неспособной доказать высказывания, которые из них следуют. Следует отметить, что алгоритм прямого логического вывода не подвержен этой проблеме: сразу после вывода фактов `path(a, b)`, `path(b, c)` и `path(a, c)` процесс прямого логического вывода останавливается.

Обратный логический вывод с поиском в глубину сталкивается также с проблемами, обусловленными излишними вычислениями. Например, при поиске пути от узла  $A_1$  к узлу  $J_4$  на рис. 9.5, *b* система Prolog выполняет 877 этапов логического вывода, большинство из которых связано с поиском всех возможных путей к узлам, не позволяющим достичь цели. Такая проблема аналогична проблеме повторяющихся состоя-

ний, которая описывалась в главе 3. Общее количество этапов логического вывода может определяться экспоненциальной зависимостью от количества базовых фактов, вырабатываемых в процессе вывода. А если бы вместо этого применялся прямой логический вывод, то можно было бы ограничиться выработкой, самое большое,  $n^2$  фактов  $\text{path}(X, Y)$ , связывающих  $n$  узлов. При этом для решения задачи, приведенной на рис. 9.5, б, потребовалось бы только 62 этапа логического вывода.

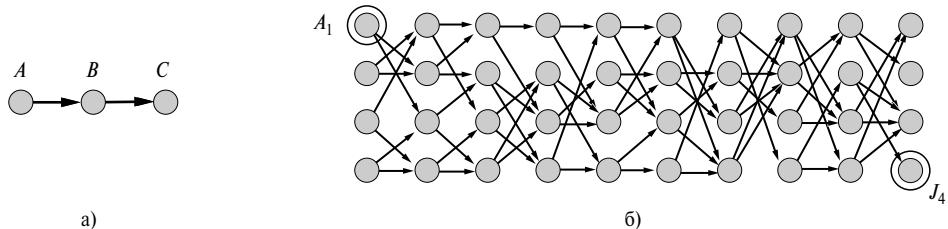


Рис. 9.5. Иллюстрация недостатков языка Prolog: поиск пути от A до C может привести систему Prolog к созданию бесконечного цикла (а); граф, в котором каждый узел связан с двумя случайно выбираемыми преемниками на следующем уровне; для поиска пути от  $A_1$  до  $J_4$  требуется 877 этапов логического вывода (б)

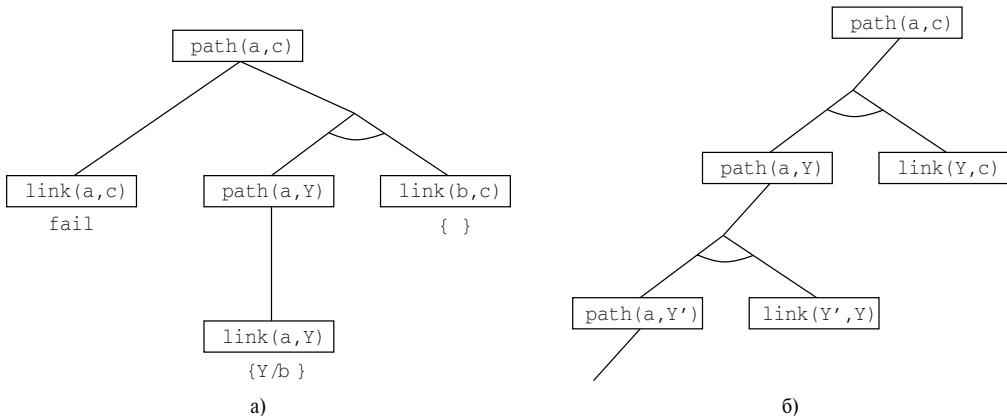


Рис. 9.6. Иллюстрация того, что работа программы Prolog зависит от расположения взаимосвязанных выражений: успешное доказательство того, что путь от A до C существует (а); бесконечное дерево доказательства, формируемое из-за того, что выражения находятся в "неправильном" порядке (б)

Применение прямого логического вывода при решении задач поиска в графе представляет собой пример **динамического программирования**, в котором решения подзадач формируются инкрементно из решений меньших подзадач и кэшируются для предотвращения повторного вычисления. Аналогичный эффект в системе обратного логического вывода может быть достигнут с помощью **запоминания** (memoization), т.е. кэширования решений, найденных для подцелей, по мере их обнаружения, а затем повторного применения этих решений после очередного появления той же подцели, вместо повторения предыдущих вычислений. Этот подход применяется в системах **табулированного логического программирования** (tabled logic programming), в которых для реализации метода запоминания используются

эффективные механизмы сохранения и выборки. В табулированном логическом программировании объединяется целенаправленность обратного логического вывода с эффективностью динамического программирования, присущей прямому логическому выводу. Кроме того, эти системы являются полными применительно к программам в формате Datalog, а это означает, что программисту приходится меньше беспокоиться о бесконечных циклах.

### Логическое программирование в ограничениях

В приведенном выше описании прямого логического вывода (раздел 9.3) было показано, как можно представить задачи удовлетворения ограничений (Constraint Satisfaction Problem — CSP) в виде определенных выражений. Стандартный язык Prolog позволяет решать подобные задачи точно таким же способом, как и алгоритм поиска с возвратами, приведенный в листинге 5.1.

Поскольку поиск с возвратами предусматривает полный перебор областей определения переменных, он может применяться только для решения задач CSP с **конечными областями определения**. В терминах Prolog это можно перефразировать таким образом, что для любой цели с несвязанными переменными должно существовать конечное количество решений. (Например, цель `diff(q, sa)`, которая определяет, что штаты Квинсленд и Южная Австралия должны быть окрашены в разные цвета, имеет шесть решений, если допускается применение трех цветов.) Задачи CSP с бесконечными областями определения (например, с целочисленными или вещественными переменными) требуют применения совсем других алгоритмов, таких как распространение пределов или линейное программирование.

Приведенное ниже выражение выполняется успешно, если подставленные в него три числа удовлетворяют неравенству треугольника.

```
triangle(X, Y, Z) :-  
    X >= 0, Y >= 0, Z >= 0, X+Y >= Z, Y+Z >= X, X+Z >= Y.
```

Если системе Prolog передан запрос `triangle(3, 4, 5)`, он будет выполнен безусловно. С другой стороны, после передачи запроса `triangle(3, 4, Z)` решение не будет найдено, поскольку подцель `Z >= 0` не может быть обработана системой Prolog. Возникающая при этом сложность состоит в том, что переменные в системе Prolog должны находиться только в одном из двух состояний: несвязанные или связанные с конкретным термом.

Связывание переменной с конкретным термом может рассматриваться как крайняя форма ограничения, а именно как ограничение равенства.  **Логическое программирование в ограничениях** (Constraint Logic Programming — CLP) позволяет ограничивать, а не связывать переменные. Решением для программы в логике ограничений является наиболее конкретное множество ограничений, налагаемых на переменные запроса, которое может быть определено с помощью базы знаний. Например, решением запроса `triangle(3, 4, Z)` является ограничение  $7 \geq Z \geq 1$ . Программы в стандартной логике представляют собой частный случай программ CLP, в которых ограничения решения должны быть не ограничениями сравнения, а ограничениями равенства, т.е. связываниями.

Системы CLP включают различные алгоритмы решения задач с ограничениями для таких вариантов ограничений, которые разрешены к использованию в языке.

Например, система, позволяющая использовать линейные неравенства с переменными, имеющими вещественные значения, может включать алгоритм линейного программирования для решения этих ограничений. Кроме того, в системах CLP принят гораздо более гибкий подход к решению запросов стандартного логического программирования. Например, вместо использования поиска в глубину, слева направо, с возвратами, в них может применяться любой из более эффективных алгоритмов, описанных в главе 5, включая эвристическое упорядочение конъюнктов, обратный переход, определение условий формирования множества отсечения и т.д. Поэтому в системах CLP сочетаются элементы алгоритмов удовлетворения ограничений, логического программирования и дедуктивных баз данных.

Кроме того, системы CLP позволяют воспользоваться преимуществами различных методов оптимизации поиска в задачах CSP, описанных в главе 5, таких как упорядочение переменных и значений, предварительная проверка и интеллектуальный возврат. В частности, разработаны проекты нескольких систем, позволяющих программисту получить больший контроль над порядком поиска для логического вывода. Например, язык MRS [539], [1321] позволяет программисту-пользователю записывать **метаправила** для определения того, какие конъюнкты должны быть опробованы в первую очередь. Например, пользователь может сформулировать правило с указанием, что в первую очередь следует пытаться достичь цели с наименьшим количеством переменных, или оформить характерные для проблемной области правила, касающиеся конкретных предикатов.

## 9.5. РЕЗОЛЮЦИЯ

Последнее из трех рассматриваемых в данной главе семейств логических систем основано на **результатии**. Как было показано в главе 7, пропозициональная резолюция — это полная процедура логического вывода для пропозициональной логики на основе опровержения. В этом разделе будет показано, как распространить резолюцию на логику первого порядка.

Проблема существования полных процедур доказательства всегда является предметом непосредственного внимания математиков. Если бы удалось найти полную процедуру доказательства для математических утверждений, это повлекло бы за собой два последствия: во-первых, вывод всех заключений мог бы осуществляться механически; во-вторых, всю математику можно было бы построить как логическое следствие некоторого множества фундаментальных аксиом. Поэтому вопрос о полноте доказательства стал в XX веке предметом наиболее важных математических работ. В 1930 году немецкий математик Курт Гёдель доказал первую **теорему о полноте** для логики первого порядка, согласно которой любое высказывание, являющееся следствием заданных аксиом, имеет конечное доказательство. (Но никакая действительно применимая на практике процедура доказательства не была найдена до тех пор, пока Дж.Э. Робинсон не опубликовал в 1965 году алгоритм резолюции.) В 1931 году Гёдель доказал еще более знаменитую **теорему о неполноте**. В этой теореме утверждается, что любая логическая система, которая включает принцип индукции (а без этого принципа удается построить лишь очень малую часть дискретной математики), обязательно является неполной. Поэтому существуют такие высказывания, которые следуют из заданных аксиом, но в рамках данной логиче-

ской системы для них невозможно найти конечное доказательство. Иголка действительно может быть в метафорическом стоге сена, но ни одна процедура не позволяет гарантировать, что она будет найдена.

Несмотря на то, что теорема Гёделя о неполноте устанавливает определенные пределы, программы автоматического доказательства теорем на основе резолюции широко применялись и применяются для обоснования математических теорем, включая несколько таких теорем, для которых до сих пор не было известно доказательство. Кроме того, программы автоматического доказательства теорем успешно использовались для проверки проектов аппаратных средств, формирования логически правильных программ, а также во многих других приложениях.

### Конъюнктивная нормальная форма для логики первого порядка

Как и в случае пропозициональной логики, для резолюции в логике первого порядка требуется, чтобы высказывания находились в **конъюнктивной нормальной форме** (Conjunctive Normal Form — CNF), т.е. представляли собой конъюнкцию выражений, в которой каждое выражение представляет собой дизъюнкцию литералов<sup>6</sup>. Литералы могут содержать переменные, на которые, согласно принятому предположению, распространяется квантор всеобщности. Например, высказывание

$$\forall x \text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$$

принимает в форме CNF следующий вид:

$$\neg \text{American}(x) \vee \neg \text{Weapon}(y) \vee \neg \text{Sells}(x, y, z) \vee \neg \text{Hostile}(z) \vee \text{Criminal}(x)$$

 *Каждое высказывание в логике первого порядка может быть преобразовано в эквивалентное с точки зрения логического вывода высказывание CNF.* В частности, высказывание CNF является невыполнимым тогда и только тогда, когда невыполнимо первоначальное высказывание, поэтому мы получаем основу для формирования доказательств от противного с помощью высказываний CNF.

Процедура преобразования любого высказывания в форму CNF весьма подобна процедуре, применяемой в пропозициональной логике, которая показана на с. 308. Принципиальное различие связано с необходимостью устранения квантов существования. Проиллюстрируем эту процедуру на примере преобразования в форму CNF высказывания “Каждого, кто любит всех животных, кто-то любит”, или

$$\forall x [\forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y \text{Loves}(y, x)]$$

Ниже приведены этапы этого преобразования.

- **Устранение импликаций:**

$$\forall x [\neg \forall y \neg \text{Animal}(y) \vee \text{Loves}(x, y)] \vee [\exists y \text{Loves}(y, x)]$$

- **Перемещение связок  $\neg$  внутрь выражений.** Кроме обычных правил для отрицаемых связок, нам нужны правила для отрицаемых квантов. Поэтому получаем следующие правила:

---

<sup>6</sup> Как показано в упр. 7.12, любое выражение может быть представлено как импликация с конъюнкцией атомов слева и дизъюнкцией атомов справа. Эта форма, иногда называемая **формой Ковалевского**, при использовании записи с символом импликации, ориентированным справа налево [851], часто бывает намного более удобной для чтения по сравнению с другими формами.

$\neg\forall x p$  принимает вид  $\exists x \neg p$   
 $\neg\exists x p$  принимает вид  $\forall x \neg p$

Рассматриваемое высказывание проходит через такие преобразования:

$\forall x [\exists y \neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y Loves(y, x)]$   
 $\forall x [\exists y \neg\neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y Loves(y, x)]$   
 $\forall x [\exists y Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y Loves(y, x)]$

Обратите внимание на то, что квантор всеобщности ( $\forall y$ ) в предпосылке импликации стал квантором существования. Теперь это высказывание приобрело такое прочтение: “Либо существует какое-то животное, которого  $x$  не любит, либо (если это утверждение не является истинным) кто-то любит  $x$ ”. Очевидно, что смысл первоначального высказывания был сохранен.

- **Стандартизация переменных.** В высказываниях наподобие  $(\forall x P(x)) \vee (\exists x Q(x))$ , в которых дважды используется одно и то же имя переменной, изменим имя одной из переменных. Это позволит в дальнейшем избежать путаницы после того, как будут удалены кванторы. Поэтому получим следующее:

$\forall x [\exists y Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z Loves(z, x)]$

- **Сколемизация.**  **Сколемизация** — это процесс устранения кванторов существования путем их удаления. В данном простом случае этот процесс подобен применению правила конкретизации с помощью квантора существования, приведенного в разделе 9.1: преобразовать  $\exists x P(x)$  в  $P(A)$ , где  $A$  — новая константа. Однако, если это правило будет непосредственно применено к высказыванию, рассматриваемому в качестве примера, то будет получено следующее высказывание:

$\forall x [Animal(A) \wedge \neg Loves(x, A)] \vee Loves(B, x)$

которое имеет полностью неправильный смысл: в нем утверждается, что каждый либо не способен любить какое-то конкретное животное  $A$ , либо его любит некоторая конкретная сущность  $B$ . В действительности первоначальное высказывание позволяет каждому человеку не быть способным любить какое-то другое животное или быть любимым другим человеком. Поэтому желательно, чтобы сущности, определяемые в процессе сколемизации, зависели от  $x$ :

$\forall x [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$

где  $F$  и  $G$  —  **сколемовские функции**. Общее правило состоит в том, что все параметры сколемовской функции должны быть переменными, на которые распространяются кванторы всеобщности, в область действия которых попадает соответствующий квантор существования. Как и при использовании конкретизации с помощью квантора существования, сколемизированное высказывание является выполнимым тогда и только тогда, когда выполнимо первоначальное высказывание.

- **Удаление кванторов всеобщности.** В данный момент на все оставшиеся переменные должны распространяться кванторы всеобщности. Кроме того, данное высказывание эквивалентно тому, в котором все кванторы всеобщности перенесены влево. Поэтому кванторы всеобщности могут быть удалены следующим образом:

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

- **Распределение связки  $\vee$  по  $\wedge$ :**

$$[Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$

На этом этапе может также потребоваться выполнить раскрытие скобок во вложенных конъюнкциях и дизъюнкциях.

Теперь рассматриваемое высказывание находится в форме CNF и состоит из двух выражений. Оно полностью недоступно для восприятия. (Помочь его понять может такое пояснение, что сколемовская функция  $F(x)$  указывает на животное, которое потенциально может быть нелюбимым лицом  $x$ , а  $G(x)$  указывает на кого-то, кто может любить лицо  $x$ .) К счастью, людям редко приходится изучать высказывания в форме CNF, поскольку показанный выше процесс преобразования может быть легко автоматизирован.

### Правило логического вывода с помощью резолюции

Правило резолюции для выражений в логике первого порядка представляет собой поднятую версию правила резолюции для пропозициональной логики, приведенного на с. 307. Два выражения, которые, согласно принятому предположению, должны быть стандартизованными таким образом, чтобы в них не было общих переменных, могут быть подвергнуты операции резолюции, если они содержат взаимно дополнительные литералы. Пропозициональные литералы являются взаимно дополнительными, если один из них представляет собой отрицание другого, а литералы в логике первого порядка являются взаимно дополнительными, если один из них унифицируется с отрицанием другого. Поэтому имеет место следующее правило:

$$\frac{\ell_1 \vee \dots \vee \ell_k, m_1 \vee \dots \vee m_n}{\text{Subst}(\theta, \ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

где  $\text{Unify}(\ell_i, \neg m_j) = \theta$ . Например, можно применить операцию резолюции к следующим двум выражениям:

$$[Animal(F(x)) \vee Loves(G(x), x)] \text{ и } [\neg Loves(u, v) \vee \neg Kills(u, v)]$$

путем устранения взаимно дополнительных литералов  $Loves(G(x), x)$  и  $\neg Loves(u, v)$  с помощью унификатора  $\theta = \{u/G(x), v/x\}$  для получения следующего выражения, называемого **резольвентой**:

$$[Animal(F(x)) \vee \neg Kills(G(x), x)]$$

Только что приведенное правило называется правилом **бинарной резолюции**, поскольку в нем происходит удаление с помощью резолюции двух и только двух взаимно дополнительных литералов. Но правило бинарной резолюции, отдельно взятое, не позволяет получить полную процедуру логического вывода. С другой стороны, правило полной резолюции позволяет удалять в каждом выражении подмножества литералов, которые являются унифицируемыми. Альтернативный подход состоит в том, чтобы распространить **операцию факторизации** (удаления избыточных литералов) на логику первого порядка. В пропозициональной факторизации два литерала сводятся к одному, если они являются идентичными, а в факторизации первого порядка два литерала сводятся к одному, если они являются унифицируемыми.

Унификатор должен быть применен ко всему выражению. Сочетание бинарной резолюции и факторизации позволяет создать полную процедуру логического вывода.

### Примеры доказательств

При использовании резолюции доказательство того, что  $KB \models \alpha$  (из базы знаний следует высказывание  $\alpha$ ) осуществляется путем доказательства невыполнимости выражения  $KB \wedge \neg\alpha$ , т.е. путем получения пустого выражения. Алгоритмический подход, применяемый в логике первого порядка, идентичен подходу в пропозициональной логике, который показан в листинге 7.5, поэтому мы не будем здесь его повторять. Вместо этого приведем два примера доказательства. Первым из них является пример доказательства преступления, описанного в разделе 9.3. Соответствующие высказывания, преобразованные в форму CNF, выглядят следующим образом:

```

¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x, y, z) ∨ ¬Hostile(z) ∨
Criminal(x)
¬Missile(x) ∨ ¬Owns(Nono, x) ∨ Sells(West, x, Nono)
¬Enemy(x, America) ∨ Hostile(x)
¬Missile(x) ∨ Weapon(x)
Owns(Nono, M1)
Missile(M1)
American(West)
Enemy(Nono, America)

```

В число этих высказываний должна быть включена также отрицаемая цель  $\neg\text{Criminal}(West)$ . Процедура доказательства по методу резолюции показана на рис. 9.7. Обратите внимание на его структуру: единственный “хребет” начинается с целевого выражения, и операция резолюции применяется к выражениям из базы знаний до тех пор, пока не образуется пустое выражение. В этом состоит характерная особенность применения метода резолюции к базам знаний, представленным в виде хорновских выражений. В действительности выражения, расположенные вдоль главного хребта, точно соответствуют последовательным значениям целевых переменных в алгоритме обратного логического вывода, приведенном в листинге 9.3. Это связано с тем, что для резолюции всегда выбирается выражение, положительный литерал которого унифицируется с самым левым литералом “текущего” выражения в хребте; именно это происходит при обратном логическом выводе. Таким образом, обратный логический вывод в действительности представляет собой просто частный случай резолюции, в котором применяется конкретная стратегия управления для определения того, какая операция резолюции должна быть выполнена в следующую очередь.

В рассматриваемом здесь втором примере используется сколемизация и применяются выражения, которые не являются определенными. Это приводит к созданию немного более сложной структуры доказательства. На естественном языке эта задача формулируется, как описано ниже.

Каждого, кто любит всех животных, кто-то любит.  
Любого, кто убивает животных, никто не любит.  
Джек любит всех животных.  
Кота по имени Тунец убил либо Джек, либо Любопытство.  
Действительно ли этого кота убило Любопытство?

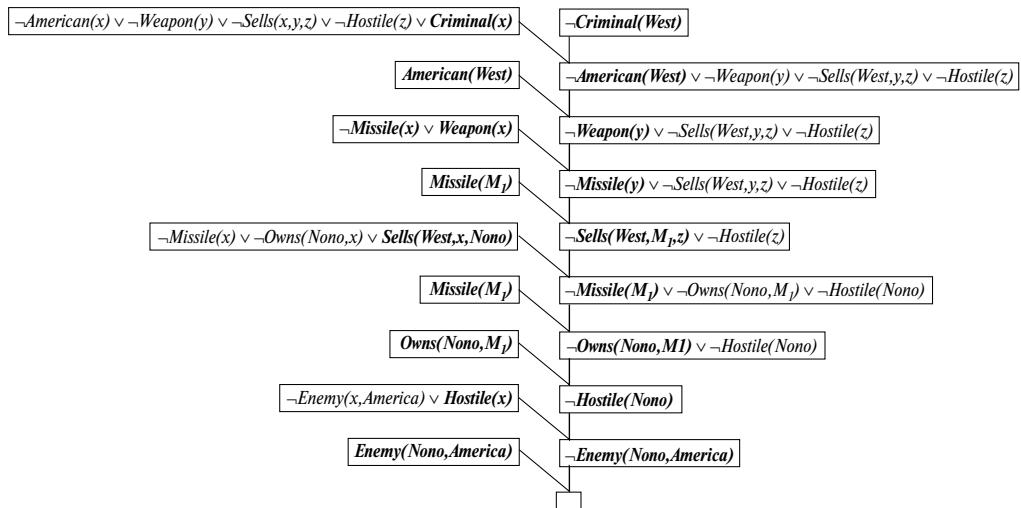


Рис. 9.7. Процедура доказательства с помощью резолюции того, что полковник Уэст совершил преступление

Вначале представим в логике первого порядка первоначальные высказывания, некоторые фоновые знания и отрицаемую цель  $G$ :

- A.  $\forall x [\forall y Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y Loves(y,x)]$
- B.  $\forall x [\exists y Animal(y) \wedge Kills(x,y)] \Rightarrow [\forall z \neg Loves(z,x)]$
- C.  $\forall x Animal(x) \Rightarrow Loves(Jack,x)$
- D.  $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
- E.  $Cat(Tuna)$
- F.  $\forall x Cat(x) \Rightarrow Animal(x)$
- $\neg G. \neg Kills(Curiosity,Tuna)$

Затем применим процедуру преобразования, чтобы преобразовать каждое высказывание в форму CNF:

- A1.  $Animal(F(x)) \vee Loves(G(x),x)$
- A2.  $\neg Loves(x,F(x)) \vee Loves(G(x),x)$
- B.  $\neg Animal(y) \vee \neg Kills(x,y) \vee \neg Loves(z,x)$
- C.  $\neg Animal(x) \vee Loves(Jack,x)$
- D.  $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
- E.  $Cat(Tuna)$
- F.  $\neg Cat(x) \vee Animal(x)$
- $\neg G. \neg Kills(Curiosity,Tuna)$

Доказательство с помощью метода резолюции того, что кота убило Любопытство, приведено на рис. 9.8. На естественном языке это доказательство может быть перефразировано, как показано ниже.

Предположим, что кота Тунца убило не Любопытство. Мы знаем, что это сделал либо Джек, либо Любопытство; в таком случае это должен был сделать Джек. Итак, Тунец — кот, а коты — животные, поэтому Тунец — животное. Любого, кто убивает животное, никто не любит, поэтому мы делаем вывод, что никто не любит Джека. С другой стороны, Джек любит всех животных, поэтому кто-то его любит; таким образом, возникает противоречие. Это означает, что кота убило Любопытство.

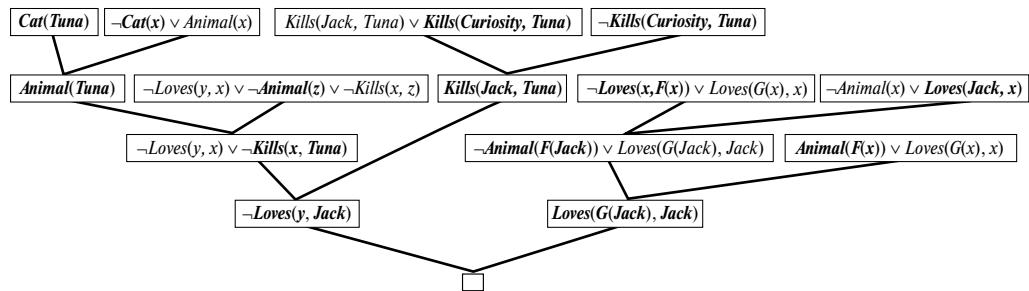


Рис. 9.8. Процедура доказательства с помощью резолюции того, что кота убило Любопытство. Обратите внимание на то, что при выводе выражения  $\text{Loves}(G(\text{Jack}), \text{Jack})$  использовалась факторизация

Такое доказательство действительно отвечает на вопрос “Действительно ли этого кота убило Любопытство?”, но часто требуется найти ответ на более общие вопросы, такие как “Кто убил кота?” Резолюция позволяет это сделать, но для получения ответа требует немного больше работы. Данная цель может быть представлена в виде выражения  $\exists w \text{ } \text{Kills}(w, \text{Tuna})$ , которое после его отрицания принимает в форме CNF вид  $\neg \text{Kills}(w, \text{Tuna})$ . Повторяя доказательство, показанное на рис. 9.8, с новой отрицаемой целью, мы получим аналогичное дерево доказательства, но с подстановкой  $\{w / \text{Curiosity}\}$  в одном из этапов. Поэтому в данном случае для поиска того, кто убил кота, достаточно проследить за связываниями, которые применяются к переменным запроса в этом доказательстве.

К сожалению, в методе резолюции могут вырабатываться **неконструктивные доказательства** для существующих целей. Например, в выражении  $\neg \text{Kills}(w, \text{Tuna})$  после применения операции резолюции удаляется взаимно дополнительный литерал, входящий в состав выражения  $\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})$ , с получением выражения  $\text{Kills}(\text{Jack}, \text{Tuna})$ , к которому снова применяется операция резолюции с использованием выражения  $\neg \text{Kills}(w, \text{Tuna})$ , что приводит к получению пустого выражения. Обратите внимание на то, что в этом доказательстве переменная  $w$  имеет два различных связывания, а правило резолюции сообщает нам, что кто-то действительно убил кота Тунца — либо Джек, либо Любопытство. Но в этом для нас нет ничего нового! Одно из решений данной проблемы состоит в том, чтобы регламентировать допустимые этапы резолюции так, чтобы переменные запроса могли быть связанными только один раз в каждом конкретном доказательстве; в таком случае можно будет предусмотреть применение перехода с возвратом по всем возможным связываниям. Еще одно решение состоит в добавлении специального **литерала ответа** к отрицаемой цели, которая принимает вид  $\neg \text{Kills}(w, \text{Tuna}) \vee \text{Answer}(w)$ . Теперь процесс резолюции вырабатывает ответ каждый раз, когда формируется выражение, содержащее только единственный литерал ответа. Для доказательства, приведенного на рис. 9.8, таковым является выражение  $\text{Answer}(\text{Curiosity})$ . Неконструктивное доказательство привело бы к выработке выражения  $\text{Answer}(\text{Curiosity}) \vee \text{Answer}(\text{Jack})$ , которое не может рассматриваться как ответ и отбрасывается.

## Полнота резолюции

В настоящем разделе приведено доказательство полноты резолюции. Это доказательство может пропустить без ущерба для дальнейшего понимания текста любой читатель, который готов принять его на веру.

Мы покажем, что резолюция обеспечивает **полноту опровержения** (refutation completeness), а это означает, что если множество высказываний является невыполнимым, то резолюция всегда позволяет прийти к противоречию. Резолюцию нельзя использовать для выработки всех логических следствий из множества высказываний, но она может применяться для подтверждения того, что данное конкретное высказывание следует из множества высказываний. Поэтому резолюция может служить для поиска всех ответов на данный конкретный вопрос с помощью метода отрицаемой цели, который был описан выше в настоящей главе.

Примем как истинное такое утверждение, что любое высказывание в логике первого порядка (без использования равенства) может быть перезаписано в виде множества выражений в форме CNF. Это можно доказать по индукции на основе анализа формы высказывания, применяя в качестве базового случая атомарные высказывания [336]. Поэтому наша цель состоит в том, чтобы доказать следующее: **если  $S$  — невыполнимое множество выражений, то применение к  $S$  конечного количества этапов резолюции приведет к противоречию.**

Схема нашего доказательства повторяет первоначальное доказательство, приведенное Робинсоном, с некоторыми упрощениями, которые были внесены Гензеретом и Нильссоном [537]. Основная структура этого доказательства показана на рис. 9.9; оно осуществляется, как описано ниже.

Любое множество предложений  $S$  представимо  
в форме импликационных выражений

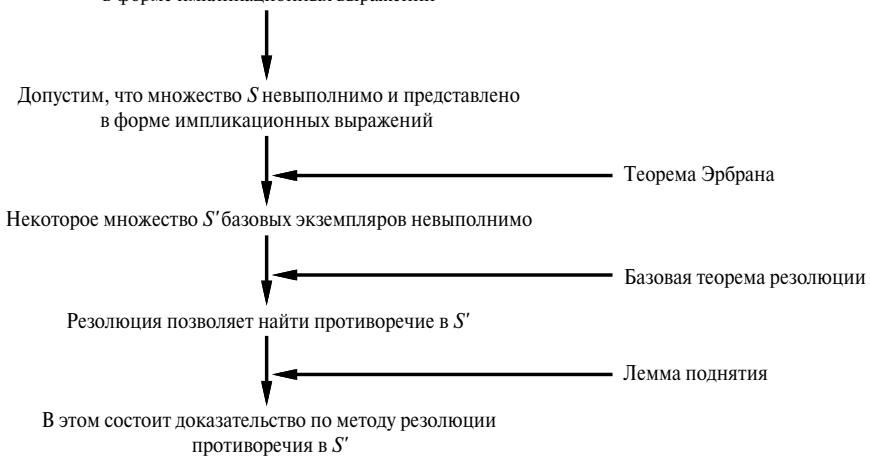


Рис. 9.9. Структура доказательства полноты резолюции

1. Вначале отметим, что если множество выражений  $S$  невыполнимо, то существует такое конкретное множество базовых экземпляров выражений  $S$ , что это множество также невыполнимо (теорема Эрбрана).

2. Затем прибегнем к **базовой теореме резолюции** (ground resolution theorem), приведенной в главе 7, в которой утверждается, что пропозициональная резолюция является полной для базовых высказываний.
3. После этого воспользуемся **леммой поднятия**, чтобы показать, что для любого доказательства по методу пропозициональной резолюции, в котором применяется множество базовых высказываний, существует соответствующее доказательство по методу резолюции первого порядка с использованием высказываний в логике первого порядка, из которых были получены базовые высказывания.

### ТЕОРЕМА ГЁДЕЛЯ О НЕПОЛНОТЕ

Немного дополнив язык логики первого порядка для обеспечения возможности применять **схему математической индукции** в арифметике, Гёдель сумел показать в своей **теореме о неполноте**, что существуют истинные арифметические высказывания, которые не могут быть доказаны.

Полное доказательство этой теоремы о неполноте немного выходит за рамки настоящей книги, поскольку в своем непосредственном виде оно занимает не меньше 30 страниц, но мы здесь приведем его набросок. Начнем с логической теории чисел. В этой теории существует единственная константа, 0, и единственная функция,  $S$  (функция определения преемника). В намеченной модели интерпретации этой теории  $S(0)$  обозначает 1,  $S(S(0))$  обозначает 2 и т.д., поэтому в рассматриваемом языке имеются имена для всех натуральных чисел. Кроме того, словарь языка включает функциональные символы +,  $\times$  и  $Expt$  (возведение в степень), а также обычное множество логических связок и кванторов. Прежде всего, следует отметить, что множество высказываний, которые могут быть записаны на этом языке, может быть пронумеровано. (Для этого достаточно представить себе, что определен алфавитный порядок символов, а затем в алфавитном порядке расположено каждое из множеств высказываний с длиной 1, 2 и т.д.) Затем можно обозначить каждое высказывание  $\alpha$  уникальным натуральным числом  $\# \alpha$  (которое называется **гёделевским номером**). Это — самый важный момент доказательства; в нем утверждается, что теория чисел включает отдельное имя для каждого из ее собственных высказываний. Аналогичным образом, с помощью гёделевского номера  $G(P)$  можно пронумеровать каждое возможное доказательство  $P$ , поскольку любое доказательство — это просто конечная последовательность высказываний.

Теперь предположим, что имеется рекурсивно перечислимое множество  $A$  высказываний, которые представляют собой истинные утверждения о натуральных числах. Напомним, что высказывания из множества  $A$  можно именовать с помощью заданного множества целых чисел, поэтому можно представить себе, что на нашем языке записывается высказывание  $\alpha(j, A)$  такого рода:

- $\forall i \ i$  — не гёделевский номер доказательства высказывания, гёделевским номером которого является  $j$ , где в доказательстве используются только предпосылки из множества  $A$ .

Затем допустим, что  $\sigma$  представляет собой высказывание  $\alpha(\#\sigma, A)$ , т.е. высказывание, в котором утверждается его собственная недоказуемость из  $A$ . (Утверждение о том, что такое высказывание всегда существует, является истинным, но его нельзя назвать полностью очевидным.)

Теперь применим следующий остроумный довод: предположим, что высказывание  $\sigma$  доказуемо из  $A$ ; в таком случае высказывание  $\sigma$  должно (поскольку в высказывании  $\sigma$  утверждается, что оно не может быть доказано). Но это означает, что имеется некоторое ложное высказывание, которое доказуемо из  $A$ , поэтому  $A$  не может состоять только из истинных высказываний, а это противоречит нашей предпосылке. Поэтому высказывание  $\sigma$  не доказуемо из  $A$ . Но именно это и утверждает о самом себе высказывание  $\sigma$ , а это означает, что  $\sigma$  — истинное высказывание.

Итак, мы доказали (и сэкономили 29 с половиной страниц), что для любого множества истинных высказываний теории чисел и, в частности, для любого множества базовых аксиом существуют другие истинные высказывания, которые не могут быть доказаны из этих аксиом. Из этого, кроме всего прочего, следует, что мы никогда не сможем доказать все теоремы математики в пределах любой конкретной системы аксиом. Очевидно, что это открытие имело для математики очень важное значение. Значимость этого открытия для искусственного интеллекта была предметом широких обсуждений, начиная с размышлений самого Гёделя. Мы вступим в эти дебаты в главе 26.

Для того чтобы выполнить первый этап доказательства, нам потребуются три новых понятия, описанных ниже.

- **Универсум Эрбрана.** Если  $S$  — множество выражений, то  $H_S$ , универсум Эрбрана для множества  $S$ , представляет собой множество всех базовых термов, которые могут быть сформированы из следующего:
  - a) функциональные символы из множества  $S$ , если они имеются;
  - b) константные символы из множества  $S$ , если они имеются; если они отсутствуют, то константный символ  $A$ .

Например, если множество  $S$  содержит только выражение  $\neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)$ , то  $H_S$  представляет собой следующее бесконечное множество базовых термов:

$$\{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), \dots\}$$

- **Насыщение.** Если  $S$  — множество выражений, а  $P$  — множество базовых термов, то  $P(S)$ , насыщение  $S$  по отношению к  $P$ , представляет собой множество всех базовых выражений, полученное путем применения всех возможных совместимых подстановок базовых термов из  $P$  вместо переменных в  $S$ .
- **База Эрбрана.** Насыщение множества выражений  $S$  по отношению к его универсуму Эрбрана называется базой Эрбрана множества  $S$  и записывается как  $H_S(S)$ . Например, если  $S$  содержит только приведенное выше выражение, то  $H_S(S)$  представляет собой следующее бесконечное множество выражений:

$$\begin{aligned} & \{\neg P(A, F(A, A)) \vee \neg Q(A, A) \vee R(A, B), \\ & \neg P(B, F(B, A)) \vee \neg Q(B, A) \vee R(B, B), \\ & \neg P(F(A, A), F(F(A, A), A)) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B), \\ & \neg P(F(A, B), F(F(A, B), A)) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B), \dots \} \end{aligned}$$

Эти определения позволяют сформулировать одну из форм **теоремы Эрбрана** [650]:

Если множество выражений  $S$  является невыполнимым, то существует конечное подмножество  $H_S(S)$ , которое также является невыполнимым.

Допустим, что  $S'$  — конечное подмножество базовых высказываний. Теперь можно прибегнуть к базовой теореме резолюции (с. 311), чтобы показать, что **резолюционное замыкание**  $RC(S')$  содержит пустое выражение. Это означает, что доведение до конца процесса пропозициональной резолюции применительно к  $S'$  приводит к противоречию.

Теперь, после определения того, что всегда существует доказательство по методу резолюции, в котором применяется некоторое конечное подмножество базы Эрбрана множества  $S$ , на следующем этапе необходимо показать, что существует доказательство по методу резолюции, в котором используются выражения из самого множества  $S$ , которые не обязательно являются базовыми выражениями. Начнем с рассмотрения одного приложения правила резолюции. Из базовой леммы Робинсона следует приведенный ниже факт.

Допустим, что  $C_1$  и  $C_2$  — два выражения без общих переменных, а  $C_1'$  и  $C_2'$  — базовые экземпляры  $C_1$  и  $C_2$ . Если  $C'$  — резольвента  $C_1'$  и  $C_2'$ , то существует выражение  $C$ , такое, что, во-первых,  $C$  — резольвента  $C_1$  и  $C_2$ , и, во-вторых,  $C'$  — базовый экземпляр  $C$ .

Это утверждение называется **леммой поднятия** (lifting lemma), поскольку оно позволяет поднять любой этап доказательства от базовых выражений к общим выражениям в логике первого порядка. Для того чтобы доказать свою основную лемму поднятия, Робинсону пришлось изобрести унификацию и определить все свойства наиболее общих унификаторов. Мы здесь не будем повторять доказательство Робинсона, а просто проиллюстрируем применение этой леммы следующим образом:

$$\begin{aligned} C_1 &= \neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B) \\ C_2 &= \neg N(G(y), z) \vee P(H(y), z) \\ C_1' &= \neg P(H(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B) \\ C_2' &= \neg N(G(B), F(H(B), A)) \vee P(H(B), F(H(B), A)) \\ C' &= \neg N(G(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B) \\ C &= \neg N(G(y), F(H(y), A)) \vee \neg Q(H(y), A) \vee R(H(y), B) \end{aligned}$$

Очевидно, что  $C'$  — действительно базовый экземпляр выражения  $C$ . Вообще говоря, для того чтобы выражения  $C_1'$  и  $C_2'$  имели какие-либо резольвенты, они должны быть получены путем предварительного применения к выражениям  $C_1$  и  $C_2$  наиболее общего унификатора для пары взаимно дополнительных литералов в  $C_1$  и  $C_2$ . Из леммы поднятия можно легко получить аналогичное, приведенное ниже утверждение о любой последовательности применений правила резолюции.

Для любого выражения  $C'$  в резолюционном замыкании множества выражений  $S'$  существует выражение  $C$  в резолюционном замыкании множества выражений  $S$ , такое, что  $C'$  — базовый экземпляр выражения  $C$  и логический вывод  $C$  имеет такую же длину, как и логический вывод  $C'$ .

Из этого факта следует, что если в резолюционном замыкании множества выражений  $S'$  появляется пустое выражение, оно должно также появиться в резолюционном замыкании множества выражений  $S$ . Это связано с тем, что пустое выражение не может быть базовым экземпляром любого другого выражения. Подведем итог: мы показали, что если множество выражений  $S$  невыполнимо, то для него существует конечная процедура логического вывода пустого выражения с помощью правила резолюции.

Поднятие способа доказательства теорем от базовых выражений к выражениям первого порядка обеспечивает огромное увеличение мощи доказательства. Это увеличение связано с тем фактом, что теперь в доказательстве первого порядка конкретизация переменных может выполняться только по мере того, как это потребуется для самого доказательства, тогда как в методах с использованием базовых выражений приходилось исследовать огромное количество произвольных конкретизаций.

### Учет отношения равенства

Ни в одном из методов логического вывода, описанных до сих пор в этой главе, не учитывалось отношение равенства. Для решения этой задачи может быть принято три различных подхода. Первый подход состоит в аксиоматизации равенства — включении в базу знаний высказываний, касающихся отношения равенства. При этом необходимо описать, что отношение равенства является рефлексивным, симметричным и транзитивным, а также сформулировать утверждение, что мы можем в любом предикате или функции заменять равные литералы равными. Таким образом, требуются три базовые аксиомы и еще по одной аксиоме для каждого предиката и функции, как показано ниже.

$$\begin{aligned} \forall x \ x = x \\ \forall x, y \ x = y \Rightarrow y = x \\ \forall x, y, z \ x = y \wedge y = z \Rightarrow x = z \\ \forall x, y \ x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \\ \forall x, y \ x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \\ \dots \\ \forall w, x, y, z \ w = y \wedge x = z \Rightarrow (F_1(w, x) = F_1(y, z)) \\ \forall w, x, y, z \ w = y \wedge x = z \Rightarrow (F_2(w, x) = F_2(y, z)) \\ \dots \end{aligned}$$

При наличии в базе знаний таких высказываний любая стандартная процедура логического вывода, такая как резолюция, позволяет решать задачи, требующие формирования рассуждений с учетом отношения равенства, например находить решения математических уравнений.

Еще один способ учета отношения равенства состоит в использовании дополнительного правила логического вывода. В простейшем правиле, правиле **демодуляции**, берется единичное выражение  $x=y$ , после чего терм  $y$  подставляется вместо любого терма, который унифицируется с термом  $x$  в каком-то другом выражении. Более формально эту идею можно представить, как описано ниже.

- **Демодуляция.** Для любых термов  $x$ ,  $y$  и  $z$ , где  $\text{Unify}(x, z) = \theta$  и  $m_n[z]$  — литерал, содержащий  $z$ , справедливо следующее:

$$\frac{x = y, m_1 \vee \dots \vee m_n[z]}{m_1 \vee \dots \vee m_n[\text{Subst}(\theta, y)]}$$

Демодуляция обычно используется для упрощения выражений с помощью коллекции утверждений, таких как  $x+0=x$ ,  $x^1=x$  и т.д. Это правило может быть также дополнено, чтобы можно было учитывать неединичные выражения, в которых появляется литерал со знаком равенства, как показано ниже.

- **¶ Парамодуляция.** Для любых термов  $x$ ,  $y$  и  $z$ , где  $\text{Unify}(x, z) = \theta$ , справедливо следующее:

$$\frac{\ell_1 \vee \dots \vee \ell_k \vee x = y, m_1 \vee \dots \vee m_n[z]}{\text{Subst}(\theta, \ell_1 \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_n[y])}$$

В отличие от демодуляции, парамодуляция позволяет получить полную процедуру логического вывода для логики первого порядка с отношением равенства.

В третьем подходе формирование логических рассуждений с учетом равенства полностью осуществляется с помощью расширенного алгоритма унификации. Это означает, что термы рассматриваются как унифицируемые, если можно доказать, что они становятся равными при некоторой подстановке; здесь выражение “можно доказать” допускает включение в определенном объеме рассуждений о равенстве. Например, термы  $1+2$  и  $2+1$  обычно не рассматриваются как унифицируемые, но алгоритм унификации, в котором известно, что  $x+y=y+x$ , способен унифицировать их с помощью пустой подстановки. **¶ Унификация с учетом равенства** (equational unification) такого рода может выполняться с помощью эффективных алгоритмов, разработанных с учетом данных конкретных используемых аксиом (коммутативность, ассоциативность и т.д.), а не с помощью явного логического вывода на основе этих аксиом. Программы автоматического доказательства теорем с использованием этого метода очень близки к системам логического программирования в ограничениях, описанным в разделе 9.4.

## Стратегии резолюции

Как известно, повторные применения правила логического вывода на основе резолюции позволяют в конечном итоге найти доказательство, если оно существует, а в этом подразделе рассматриваются стратегии, позволяющие находить доказательства не методом перебора, а более эффективно.

### Преимущественное использование единичных выражений

В этой стратегии преимущество отдается таким операциям резолюции, в которых одним из высказываний является единственный литерал (известный также как **единичное выражение** — unit clause). В основе этой стратегии лежит такая идея, что если осуществляются попытки получения пустого выражения, то может оказаться целесообразным отдавать предпочтение таким операциям логического вывода, в которых вырабатываются более короткие выражения. Применение операции резолюции к единичному высказыванию (такому как  $P$ ) в сочетании с любым другим высказыванием (таким как  $\neg P \vee \neg Q \vee R$ ) всегда приводит к получению выражения (в данном случае  $\neg Q \vee R$ ), более короткого, чем это другое высказывание. Когда стратегия

гия с преимущественным использованием единичных выражений была впервые опробована в пропозициональном логическом выводе в 1964 году, она привела к резкому ускорению работы, обеспечив возможность доказывать теоремы, с которыми не удавалось справиться без использования этого метода предпочтения. Тем не менее метод предпочтения единичных выражений, отдельно взятый, не позволяет уменьшить коэффициент ветвления в задачах средних размеров до такой степени, чтобы можно было обеспечить возможность их решения с помощью резолюции. Несмотря на это, он представляет собой полезный эвристический метод, который может успешно использоваться в сочетании с другими стратегиями.

❖ **Единичная резолюция** (*unit resolution*) — это ограниченная форма резолюции, в которой на каждом этапе резолюции должно участвовать единичное выражение. В общем случае метод единичной резолюции является неполным, но становится полным при его применении к хорновским базам знаний. Процесс доказательства по методу единичной резолюции применительно к хорновским базам знаний напоминает прямой логический вывод.

### **Множество поддержки**

Применение метода предпочтений, в котором в первую очередь осуществляется попытка выполнить определенные операции резолюции, вполне оправдано, но, вообще говоря, более эффективный метод может быть основан на том, что следует попытаться полностью устранить некоторые потенциальные этапы резолюции. В стратегии с использованием множества поддержки выполняется именно это. Применение данной стратегии начинается с выявления подмножества высказываний, называемого ❖ **множеством поддержки** (*set of support*). На каждом этапе резолюции высказывание из множества поддержки комбинируется с другим высказыванием, а резольвента добавляется к множеству поддержки. Если множество поддержки является небольшим по сравнению со всей базой знаний, это позволяет резко сократить пространство поиска.

При использовании этого подхода необходимо соблюдать осторожность, поскольку при неправильном выборе множества поддержки алгоритм может стать неполным. Однако, если множество поддержки  $S$  будет выбрано так, чтобы оставшиеся высказывания, вместе взятые, оставались выполнимыми, то резолюция с помощью множества поддержки становится полной. Общепринятый подход, основанный на предположении, что первоначальная база знаний является непротиворечивой, состоит в том, чтобы в качестве множества поддержки применялся отрицаемый запрос. (В конце концов, если база знаний не является непротиворечивой, то сам факт, что запрос является следствием из нее, становится избыточным, поскольку из противоречия можно доказать все, что угодно.) Стратегия с использованием множества поддержки имеет дополнительное преимущество в том, что в ней часторабатываются деревья доказательства, легко доступные для понимания людей, поскольку само формирование доказательства осуществляется целенаправленно.

### **Резолюция с входными высказываниями**

В стратегии ❖ **резолюции с входными высказываниями** (*input resolution*) на каждом этапе резолюции комбинируется одно из входных высказываний (из базы знаний или запроса) с некоторым другим высказыванием. В доказательстве, показанном на рис. 9.7, использовались только этапы резолюции с входными высказываниями и поэтому дерево доказательства имело характерную форму в виде единого “хребта”

с отдельными высказываниями, комбинирующими с этим хребтом. Очевидно, что пространство деревьев доказательства такой формы меньше по сравнению с пространством всех возможных графов доказательства. В хорновских базах знаний как своего рода стратегия резолюции с входными высказываниями может рассматриваться правило отделения, поскольку при использовании этого правила некоторая импликация из первоначальной базы знаний комбинируется с некоторыми другими высказываниями. Таким образом, нет ничего удивительного в том, что метод резолюции с входными высказываниями является полным применительно к базам знаний, которые находятся в хорновской форме, но в общем случае он неполон. Стратегия **линейной резолюции** (linear resolution) представляет собой небольшое обобщение, в котором допускается применять в одной операции резолюции высказывания  $P$  и  $Q$ , если  $P$  находится в первоначальной базе знаний или  $P$  является предком  $Q$  в дереве доказательства. Метод линейной резолюции является полным.

### Обобщение

В методе **обобщения** (subsumption) устраняются все высказывания, которые обобщаются некоторым существующим высказыванием из базы знаний (т.е. являются более конкретными по сравнению с ним). Например, если в базе знаний есть высказывание  $P(x)$ , то нет смысла вводить в нее высказывание  $P(A)$  и еще меньше смысла вводить  $P(A) \vee Q(B)$ . Обобщение позволяет поддерживать небольшие размеры базы знаний и тем самым ограничивать размеры пространства поиска.

## Средства автоматического доказательства теорем

Средства автоматического доказательства теорем (называемые также *средствами автоматизированного формирования рассуждений*) отличаются от языков логического программирования в двух отношениях. Во-первых, большинство языков логического программирования поддерживает только хорновские выражения, тогда как средства автоматического доказательства теорем поддерживают полную логику первого порядка. Во-вторых, в программах на таком типичном языке логического программирования, как Prolog, переплетаются логика и управление. Например, выбор программистом выражения  $A :- B, C$  вместо  $A :- C, B$  может повлиять на выполнение программы. С другой стороны, в большинстве средств автоматического доказательства теорем синтаксическая форма, выбранная для высказываний, не влияет на результаты. Для средств автоматического доказательства теорем все еще требуется управляющая информация, чтобы они могли функционировать эффективно, но эта информация обычно хранится отдельно от базы знаний, а не входит в состав самого представления знаний. Большинство исследований в области средств автоматического доказательства теорем посвящено поиску стратегий управления, которые приводят к общему повышению эффективности, а не только к увеличению быстродействия.

### Проект одного из средств автоматического доказательства теорем

В этом разделе описана программа автоматического доказательства теорем Otter (Organized Techniques for Theorem-proving and Effective Research) [1018]; в этом описании особое внимание будет уделено применяемой в ней стратегии управления. Подготавливая любую задачу для программы Otter, пользователь должен разделить знания на четыре описанные ниже части.

- Множество выражений, известное как **множество поддержки** (или *sos* — set of support), в котором определяются важные факты о данной задаче. На каждом этапе резолюции операция резолюции применяется к одному из элементов множества поддержки и к другой аксиоме, поэтому поиск сосредоточивается на множестве поддержки.
- Множество **полезных аксиом** (usable axiom), которое выходит за пределы множества поддержки. Эти аксиомы предоставляют фоновые знания о проблемной области. Определение границы между тем, что должно войти в состав задачи (и поэтому в множество *sos*) и что относится к фоновым знаниям (и поэтому должно войти в число полезных аксиом), передается на усмотрение пользователя.
- Множество уравнений, известных как **правила перезаписи** (rewrites), или **демодуляторы** (demodulators). Хотя демодуляторы представляют собой уравнения, они всегда применяются в направлении слева направо, поэтому определяют каноническую форму, в которой должны быть представлены все упрощенные термы. Например, демодулятор  $x+0=x$  указывает, что любой терм в форме  $x+0$  должен быть заменен термом  $x$ .
- Множество параметров и выражений, который определяет стратегию управления. В частности, пользователь задает эвристическую функцию для управления поиском и функцию фильтрации для устранения некоторых подшелей как не представляющих интереса.

Программа *Otter* действует по принципу постоянного применения правила резолюции к одному из элементов множества поддержки и к одной из полезных аксиом. В отличие от системы Prolog, в этой программе используется определенная форма поиска по первому наилучшему совпадению. Ее эвристическая функция измеряет “вес” каждого выражения с учетом того, что наиболее предпочтительными являются выражения с наименьшими весами. Задача точной формулировки эвристической функции возлагается на пользователя, но, вообще говоря, вес любого выражения должен коррелировать с его размером или сложностью. Единичные выражения оцениваются как имеющие наименьший вес, поэтому такой метод поиска может рассматриваться как обобщение стратегии с преимущественным использованием единичных выражений. На каждом этапе программа *Otter* перемещает выражение “с наименьшим весом” из множества поддержки в список полезных аксиом и добавляет в множество поддержки некоторые непосредственные следствия применения операции резолюции к выражению с наименьшим весом и к элементам списка полезных аксиом. Программа *Otter* останавливается, если обнаруживает противоречие или если возникает такая ситуация, что в множестве поддержки не остается больше выражений. Алгоритм работы этой программы показан более подробно в листинге 9.5.

**Листинг 9.5. Набросок структуры программы автоматического доказательства теорем *Otter*. Эвристическое управление применяется при выборе выражения “с наименьшим весом” и в функции *Filter*, которая устраняет из рассмотрения такие выражения, которые не представляют интереса**

---

```

procedure Otter(sos, usable)
  inputs: sos, множество поддержки - выражения, определяющие
         решаемую задачу (глобальная переменная)
         usable, множество фоновых знаний, которые потенциально

```

могут быть релевантными для данной задачи

```

repeat
    clause ← элемент множества sos с наименьшим весом
    переместить выражение clause из множество sos
    в множество usable
    Process(Infer(clause, usable), sos)
until sos = [] or обнаружится опровержение

function Infer(clause, usable) returns множество выражений clauses
    применить правило резолюции к выражению clause и каждому элементу
    множества usable
    возвратить полученное множество clauses после применения
    функции Filter

procedure Process(clauses, sos)
    for each clause in clauses do
        clause ← Simplify(clause)
        выполнить слияние идентичных литералов
        отбросить выражение clause, если оно представляет
        собой тавтологию
        sos ← [clause | sos]
        if clause не имеет литералов, то обнаружено опровержение
        if clause имеет один литерал, то искать
        единичное опровержение

```

### Расширение системы Prolog

Еще один способ создания средства автоматического доказательства теорем состоит в том, чтобы начать с компилятора Prolog и дополнить его в целях получения непротиворечивого и полного средства формирования рассуждений для полной логики первого порядка. Именно этот подход был принят при создании программы PTTP (Prolog Technology Theorem Prover) [1463]. Как описано ниже, программа PTTP включает пять существенных дополнений к системе Prolog, позволяющих восстановить полноту и выразительность алгоритма обратного логического вывода.

- В процедуру унификации снова вводится проверка вхождения для того, чтобы эта процедура стала непротиворечивой.
- Поиск в глубину заменяется поиском с итеративным углублением. Это позволяет добиться того, чтобы стратегия поиска стала полной, а увеличение продолжительности поиска измерялось лишь постоянной зависимостью от времени.
- Разрешается применение отрицаемых литералов (таких как  $\neg P(x)$ ). В этой реализации имеется две отдельные процедуры; в одной из них предпринимается попытка доказать  $P$ , а в другой — доказать  $\neg P$ .
- Выражение с  $n$  атомами хранится в виде  $n$  различных правил. Например, при наличии в базе знаний выражения  $A \Leftarrow B \wedge C$  должно быть также предусмотрено хранение в ней этого выражения, представленного как  $\neg B \Leftarrow C \wedge \neg A$  и как  $\neg C \Leftarrow B \wedge \neg A$ . Применение такого метода, известного под названием

❖ **блокирование** (locking), означает, что текущая цель требует унификации только с головой каждого выражения, и вместе с тем позволяет должным образом учитывать отрицание.

- Логический вывод сделан полным (даже для нехорновских выражений) путем добавления правила резолюции с линейным входным выражением: если текущая цель унифицируется с отрицанием одной из целей в стеке, то данная цель может рассматриваться как решенная. В этом состоит один из способов рассуждения от противного. Предположим, что первоначальной целью было высказывание  $P$  и что эта цель свелась в результате применения ряда этапов логического вывода к цели  $\neg P$ . Тем самым установлено, что  $\neg P \Rightarrow P$ , а это выражение логически эквивалентно  $P$ .

Несмотря на эти изменения, программа РТТР сохраняет свойства, благодаря которым обеспечивается высокое быстродействие системы Prolog. Операции унификации все еще осуществляются посредством непосредственной модификации переменных, а отмена связывания выполняется путем разгрузки контрольного стека во время возврата. Стратегия поиска все еще основана на резолюции с применением входных выражений, а это означает, что в каждой операции резолюции участвует одно из выражений, содержащихся в первоначальной формулировке задачи (а не какое-то производное выражение). Такой подход позволяет осуществить компиляцию всех выражений из первоначальной формулировки задачи.

Основным недостатком программы РТТР является то, что пользователь должен отказаться от любых попыток взять на себя управление поиском решений. Каждое правило логического вывода в этой системе используется и в его первоначальной, и в контрапозитивной форме. Это может привести к выполнению таких операций поиска, которые противоречат здравому смыслу. Например, рассмотрим следующее правило:

$$(f(x, y) = f(a, b)) \Leftarrow (x = a) \wedge (y = b)$$

Если бы оно рассматривалось как правило Prolog, то применялся бы разумный способ доказательства того, что два терма  $f$  равны. Но в системе РТТР должно быть также сформировано контрапозитивное правило:

$$(x \neq a) \Leftarrow (f(x, y) \neq f(a, b)) \wedge (y = b)$$

По-видимому, попытка доказать, что любые два терма,  $x$  и  $a$ , являются разными, привела бы к непроизводительным затратам ресурсов.

### **Применение средств автоматического доказательства теорем в качестве помощников**

До сих пор в этой книге любая система формирования рассуждений рассматривалась как независимый агент, который должен был принимать решения и действовать самостоятельно. Еще одно направление использования средств автоматического доказательства теорем состоит в том, что они должны служить в качестве помощников, предоставляя консультации, скажем, математикам. При эксплуатации подобных систем в режиме помощи математик действует в роли руководителя, очерчивая стратегию определения того, что должно быть сделано в следующую очередь, а затем передавая системе автоматического доказательства теорем просьбу проработать все детали. Это позволяет в определенной степени устранить проблему полуразрешимости, поскольку руководитель научной разработки может отменить запрос и опробовать другой подход, если поиск ответа на запрос потребовал слишком много

времени. Любая система автоматического доказательства теорем может также действовать в качестве **средства проверки доказательства**; при ее использовании в таком режиме доказательство предоставляется человеком в виде ряда довольно крупных этапов; отдельные операции логического вывода, которые требуются для того, чтобы показать, что каждый из этих этапов является непротиворечивым, определяются системой.

В частности, **сократовское средство формирования рассуждений** (socratic reasoner) представляет собой такую систему автоматического доказательства теорем, в которой функция Ask является неполной, но эта система всегда позволяет прийти к определенному решению, если ей будет задан правильный ряд вопросов. Таким образом, сократовские средства формирования рассуждений становятся хорошими помощниками, при условии, что есть руководитель, способный составить правильный ряд вызовов функции Ask. Одной из сократовских систем формирования рассуждений для математиков является Ontic [1005].

### Области практического использования средств автоматического доказательства теорем

Средства автоматического доказательства теорем позволили получить новейшие математические результаты. Программа SAM (Semi-Automated Mathematics) была первой программой, с помощью которой удалось доказать одну из лемм в теории решеток [602]. Кроме того, программа Aima позволила найти ответ на многие открытые вопросы в нескольких областях математики [1621]. Программа автоматического доказательства теорем Бойера–Мура [165] применялась и дорабатывалась в течение многих лет, и ею воспользовался Натараджан Шанкар для получения первого полностью строгого формального доказательства теоремы Гёделя о неполноте [1393]. Одним из самых строгих средств автоматического доказательства теорем является программа Otter; она использовалась для решения некоторых открытых задач в области комбинаторной логики. Наиболее известные из них касаются **алгебры Роббинса**. В 1933 году Герберт Роббинс предложил простое множество аксиом, которые, на первый взгляд, могли служить для определения булевой алгебры, но ни одного доказательства этой гипотезы найти не удавалось (несмотря на напряженную работу нескольких выдающихся математиков, включая самого Альфреда Тарского). Наконец, 10 октября 1996 года после восьми дней вычислений программа EQP (одна из версий программы Otter) нашла такое доказательство [1019].

Средства автоматического доказательства теорем могут также применяться для решения задач, связанных с **проверкой** и **синтезом** как аппаратных, так и программных средств, поскольку для обеих этих проблемных областей могут быть предусмотрены правильные варианты аксиоматизации. Поэтому исследования доказательства теорем проводятся не только в искусственном интеллекте, но и в таких областях, как проектирование аппаратных средств, языки программирования и разработка программного обеспечения. В случае программного обеспечения аксиомы определяют свойства каждого синтаксического элемента языка программирования. (Процесс формирования рассуждений о программах полностью аналогичен процессу формирования рассуждений о действиях в ситуационном исчислении.) Программный алгоритм проверяется путем демонстрации того, что его выходы соответствуют спецификациям для всех входов. Таким образом были проверены алгоритм шифрования RSA с открытым ключом и алгоритм согласования строк Бойера–Мура [166]. В случае аппаратного обеспечения аксиомы описывают способы взаи-

модействия сигналов и элементов схемы (один из примеров приведен в главе 8). Программой Aura [1610] был проверен проект 16-битового сумматора. Системы формирования логических рассуждений, предназначенные специально для проверки аппаратных средств, оказались способными проверять целые процессоры, включая свойства синхронизации этих процессоров [1455].

Формальный синтез алгоритмов был одним из первых направлений использования средств автоматического доказательства теорем, как и было намечено Корделлом Грином [591], который опирался на идеи, высказанные ранее Саймоном [1416]. Общий замысел состоит в том, что должна быть доказана теорема с утверждением, что “существует программа  $p$ , удовлетворяющая некоторой спецификации”. Если удастся ограничить это доказательство конструктивной формой, то появляется возможность извлечь из результатов доказательства требуемую программу. Хотя полностью автоматизированный ~~и~~ **дедуктивный синтез**, как стало называться это направление, еще не осуществим применительно к программированию задач общего назначения, дедуктивный синтез, управляемый вручную, оказался успешным при проектировании нескольких новейших и сложнейших алгоритмов. Кроме того, активной областью исследования является синтез программ специального назначения. В области синтеза аппаратных средств программа автоматического доказательства теорем Aura применялась для проектирования схем, оказавшихся более компактными по сравнению с разработанными во всех предыдущих проектах [1609]. Для многих проектов логических схем достаточно применить пропозициональную логику, поскольку множество интересующих высказываний является фиксированным благодаря конечным размерам множества схемных элементов. В наши дни применение пропозиционального логического вывода в аппаратном синтезе является стандартным методом, имеющим много крупномасштабных областей использования (см., например, работу Новика и др. [1149]).

Те же методы теперь начинают также применяться для проверки программного обеспечения с помощью таких систем, как программа проверки моделей Spin [672]. Например, с ее помощью была проверена до и после полета программа управления космическим аппаратом Remote Agent [633].

## 9.6. РЕЗЮМЕ

В этой главе приведен анализ логического вывода в логике первого порядка и многих алгоритмов его выполнения.

- В первом подходе используется правило логического вывода для конкретизации кванторов в целях преобразования задачи логического вывода в форму пропозициональной логики. Как правило, этот подход характеризуется очень низким быстродействием.
- Использование **унификации** для выявления подходящих подстановок для переменных позволяет устранить этап конкретизации в доказательствах первого порядка, в результате чего этот процесс становится гораздо более эффективным.
- В поднятой версии **правила отделения** унификация применяется для получения естественного и мощного правила логического вывода — **обобщенного правила**

**отделения.** В алгоритмах **прямого логического вывода и обратного логического вывода** это правило применяется к множествам определенных выражений.

- Обобщенное правило отделения является полным применительно к определенным выражениям, но проблема логического следствия остается **полуразрешимой**. Для программ **Datalog**, состоящих из определенных выражений без функций, проблема логического следствия разрешима.
- Прямой логический вывод используется в **дедуктивных базах данных**, где он может сочетаться с реляционными операциями баз данных. Он также применяется в **продукционных системах**, которые обеспечивают эффективное обновление при наличии очень больших наборов правил.
- Прямой логический вывод для программ Datalog является полным и выполняется за время, определяемое полиномиальной зависимостью.
- Обратный логический вывод используется в **системах логического программирования**, таких как **Prolog**, в которых реализована сложная технология компиляции для обеспечения очень быстрого логического вывода.
- Недостатками обратного логического вывода являются излишние этапы логического вывода и бесконечные циклы; эти недостатки можно устранить путем **запоминания**.
- Обобщенное правило логического вывода на основе **резолюции** позволяет создать полную систему доказательства для логики первого порядка с использованием баз знаний в конъюнктивной нормальной форме.
- Существует несколько стратегий сокращения пространства поиска для систем с резолюцией без потери полноты. Эффективные средства автоматического доказательства теорем на основе резолюции использовались для доказательства интересных математических теорем, а также для проверки и синтеза программных и аппаратных средств.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Логический вывод широко исследовался древнегреческими математиками. Аристотель тщательно исследовал один из типов логического вывода, называемый **силлогизмом**, который представляет собой своего рода правило логического вывода. Силлогизмы Аристотеля включали элементы логики первого порядка, такие как кванторы, но были ограничены унарными предикатами. Силлогизмы классифицировались по “фигурам” и “модусам”, в зависимости от порядка термов (которые следовало бы назвать предикатами) в высказываниях и от степени общности (которую теперь принято интерпретировать с помощью кванторов), применяемой к каждому терму, а также с учетом того, является ли каждый терм отрицаемым. Наиболее фундаментальным силлогизмом является тот, который относится к первой фигуре первого модуса:

Все  $S$  суть  $M$ .

Все  $M$  суть  $P$ .

Следовательно, все  $S$  суть  $P$ .

Аристотель пытался доказать истинность других силлогизмов, “приводя” их к силлогизмам первой фигуры. Описание того, в чем должно состоять такое “приведение”, оказалось гораздо менее точным по сравнению с описанием, в котором были охарактеризованы сами фигуры и модусы силлогизмов.

Готтлоб Фреге, который разработал полную логику первого порядка в 1879 году, основал свою систему логического вывода на большой коллекции логически правильных схем и единственном правиле логического вывода — правиле отделения (*Modus Ponens*). Фреге воспользовался тем фактом, что результат применения любого правила логического вывода в форме “Из  $P$  вывести  $Q$ ” можно моделировать путем применения к высказыванию  $P$  правила отделения наряду с логически правильной схемой  $P \Rightarrow Q$ . Такой “аксиоматический” стиль представления знаний, в котором наряду с правилом отделения использовался целый ряд логически правильных схем, был принят на вооружение после Фреге многими логиками; наиболее замечательным является то, что этот стиль использовался в основополагающей книге *Principia Mathematica* [1584].

В подходе на основе **натуральной дедукции** (natural deduction), который был введен Герхардом Генценом [541] и Станиславом Яськовским [725], основное внимание было сосредоточено не на аксиоматических системах, а на правилах логического вывода. Натуральная дедукция получила название “натуральной”, поскольку в ней не требуется преобразование в нормальную форму (неудобную для восприятия человеком), а также в связи с тем, что в ней применяются такие правила логического вывода, которые кажутся естественными для людей. Правитц [1235] посвятил описанию натуральной дедукции целую книгу. Галье [515] применил подход Гентцена для разъяснения теоретических основ автоматизированной дедукции.

Крайне важным этапом в разработке глубокого математического анализа логики первого порядка явилось изобретение формы представления в виде импликационных выражений (clausal form). Уайтхед и Рассел [1584] описали так называемые правила прохождения (rules of passage) (фактически этот термин принадлежит Эрбрану [650]), которые используются для перемещения кванторов в переднюю часть формул. Торальфом Сколемом [1424] были достаточно своевременно предложены сколемовские константы и сколемовские функции. Общая процедура сколемизации, наряду с важным понятием универсума Эрбрана, описана в [1425].

Крайне важную роль в разработке автоматизированных методов формирования рассуждений, как до, так и после введения Робинсоном правила резолюции, играет теорема Эрбрана, названная в честь французского логика Жака Эрбрана [650]. Это отражается в применяемом нами термине “универсум Эрбрана”, а не “универсум Сколема”, даже несмотря на то, что это понятие в действительности было открыто Сколемом. Кроме того, Эрбран может считаться изобретателем операции унификации. Гёдель [565], опираясь на идеи Сколема и Эрбрана, сумел показать, что для логики первого порядка имеется полная процедура доказательства. Алан Тьюринг [1518] и Алонсо Черч [255] практически одновременно продемонстрировали, используя очень разные доказательства, что задача определения общезначимости в логике первого порядка не имеет решения. В превосходной книге Эндертона [438] все эти результаты описаны в строгой, но труднодоступной для понимания манере.

Хотя Маккарти [1009] предложил использовать логику первого порядка для представления знаний и формирования рассуждений в искусственном интеллекте, первые подобные системы были разработаны логиками, заинтересованными в получе-

нии средств автоматического доказательства математических теорем. Впервые применение метода пропозиционализации и теоремы Эрбрана предложено Абрахамом Робинсоном, а Гилмор [556] написал первую программу, основанную на этом подходе. Дэвис и Патнем [336] применили форму представления в виде импликационных выражений и разработали программу, в которой предпринимались попытки поиска противоречий путем подстановки элементов универсума Эрбрана вместо переменных для получения базовых выражений, а затем поиска пропозициональных противоречивостей среди этих базовых выражений. Правиц [1234] разработал ключевую идею, позволяющую использовать для управления процессом поиска тенденцию к обнаружению пропозициональных противоречивостей и вырабатывать термы из универсума Эрбрана, только если это необходимо для определения пропозициональной противоречивости. После дальнейшей разработки этой идеи другими исследователями Дж.Э. Робинсон (который не связан родством с Абрахамом Робинсоном) пришел к созданию метода резолюции [1298]. Примерно в то же время советским исследователем С. Масловым [994], [995] на основе немного иных принципов был разработан так называемый *инверсный метод*, который характеризуется некоторыми вычислительными преимуществами над пропозиционализацией. **Метод соединения** Вольфганга Бибеля [123] может рассматриваться как расширение этого подхода.

После разработки метода резолюции исследования в области логического вывода первого порядка стали развиваться в нескольких разных направлениях. В искусственном интеллекте метод резолюции применялся для создания систем поиска ответов на вопросы Корделлом Грином и Берtramом Рафаэлем [593]. Несколько менее формальный подход был принят Карлом Хьюиттом [651]. Разработанный им язык Planner, хотя и не был полностью реализован, явился предшественником логического программирования и включал директивы для прямого и обратного логического вывода и для отрицания как недостижения цели. А подмножество этого языка, известное как Micro-Planner [1475], было реализовано и использовалось в системе понимания естественного языка Shrdlu [1601]. В ранних реализациях систем искусственного интеллекта большие усилия направлялись на разработку структур данных, которые должны были обеспечить эффективную выборку фактов; эти работы описаны в книгах по программированию для искусственного интеллекта [240], [479], [1148].

В начале 1970-х годов в искусственном интеллекте полностью утвердился метод **прямого логического вывода** как легко доступная пониманию альтернатива методу резолюции. Прямой логический вывод использовался в самых различных системах, начиная от программы автоматического доказательства геометрических теорем Невинса [1123] и заканчивая экспертной системой R1 для разработки конфигурации компьютеров VAX [1026]. Приложениянского интеллекта обычно охватывают большое количество правил, поэтому было важно разработать эффективную технологию согласования с правилами, особенно для инкрементных обновлений. Для поддержки таких приложений была разработана технология **продукционных систем**. Язык продукционных систем Ops-5 [197], [482] использовался для экспертной системы R1 и для когнитивной архитектуры Soar [880]. В языке Ops-5 был включен процесс согласования с помощью rete-алгоритма [483]. Архитектура Soar, позволяющая вырабатывать новые правила для кэширования результатов предыдущих вычислений, способна создавать очень большие множества правил; например, в системе TacAir-Soar, предназначеннной для управления тренажером, моделирующим самолет-истребитель [743], количество правил превышало один миллион. Язык

CLIPS [1626] продукционных систем на основе языка С, разработанный в NASA, обеспечивал лучшую интеграцию с другими программными, аппаратными и сенсорными системами и использовался для автоматизации космической станции и разработки нескольких военных приложений.

Большой вклад в понимание особенностей прямого логического вывода внесли также работы в области исследований, известной как **дедуктивные базы данных**. Исследования в этой области начались с симпозиума, организованного в Тулузе в 1977 году Джеком Минкером, который собрал вместе специалистов в области логического вывода и систем баз данных [514]. В опубликованном сравнительно недавно историческом обзоре [1264] сказано: “Дедуктивные системы [баз данных] были попыткой адаптировать язык Prolog, воплощающий видение мира с «малыми данными», к миру «больших данных»”. Таким образом, цель разработок в этой области состоит в объединении технологии баз данных, которая предназначена для выборки больших множеств фактов, с технологией логического вывода на основе языка Prolog, в которой обычно осуществляется выборка одновременно только одного факта. К числу работ в области дедуктивных баз данных относятся [228] и [1525].

Важная работа, выполненная Чандой и Нарелом [231], а также Ульманом [1524], привела к признанию языка Datalog в качестве стандартного языка для дедуктивных баз данных. Кроме того, стал стандартным “восходящий” логический вывод, или прямой логический вывод, отчасти потому, что данный метод позволяет избежать проблем, обусловленных незавершаемыми и избыточными вычислениями, которые возникают при обратном логическом выводе, и отчасти потому, что имеет более естественную реализацию в терминах основных операций реляционной базы данных. Разработка метода **магических множеств** для перезаписи правил Бансильоном и др. [67] позволила воспользоваться в прямом логическом выводе преимуществами цепенаправленности, свойственными обратному логическому выводу. Методы табулированного логического программирования (с. 433), вступившие в конкурентную борьбу с другими методами, заимствовали преимущества динамического программирования от прямого логического вывода.

Большой вклад в понимание сложностей логического вывода внесло сообщество пользователей дедуктивных баз данных. Чандря и Мерлин [232] впервые показали, что задача согласования единственного нерекурсивного правила (в терминологии баз данных — **конъюнктивного запроса**) может оказаться NP-трудной. Купер и Варди [870] предложили использовать понятие **сложности данных** (т.е. сложности как функции размера базы данных, в которой размер правила рассматривается как постоянный) в качестве подходящего критерия эффективности получения ответов на запросы. Готтлоб и др. [586] обсудили связь между конъюнктивными запросами и задачами удовлетворения ограничений, показав, как можно использовать способ декомпозиции гипердерева для оптимизации процесса согласования.

Как уже упоминалось выше, процедуры **обратного логического вывода**, применяемые для логического вывода, впервые появились в разработанном Хьюиттом языке Planner [651]. Но логическое программирование как таковое развивалось независимо от этого направления разработок. Ограниченная форма линейной резолюции, называемая **SL-резолюцией**, была разработана Ковалевским и Кюнером [853] на основе метода **устранения моделей** Лавленда [947]; после применения этого метода к определенным выражениям он принял вид метода **SLD-резолюции**, который предоставил возможность осуществлять интерпретацию определенных выражений как про-

грамм [849–851]. Между тем в 1972 году французский исследователь Ален Колмерор разработал и реализовал **Prolog** в целях синтаксического анализа текста на естественном языке; первоначально выражения Prolog предназначались для использования в качестве правил контекстно-свободной грамматики [285], [1311]. Основная часть теоретических основ логического программирования разработана Ковалевским в сотрудничестве с Колмерором. Семантическое определение с использованием наименьших фиксированных точек предложено Ван Эмденом и Ковалевским [1530]. Ковалевский и Коэн [274], [852] подготовили хорошие исторические обзоры истоков языка Prolog. Теоретический анализ основ языка Prolog и других языков логического программирования приведен в книге *Foundations of Logic Programming* [940].

Эффективные компиляторы Prolog главным образом основаны на модели абстрактной машины Уоррена (Warren Abstract Machine — WAM), разработанной Дэвидом Г.Д. Уорреном [1556]. Ван Рой [1536] показал, что благодаря применению дополнительных методов организации работы компилятора, таких как логический вывод типов, программы Prolog становятся способными конкурировать по быстродействию с программами С. Рассчитанный на 10 лет исследовательский проект создания компьютера пятого поколения, который был развернут в Японии в 1982 году, опирался полностью на язык Prolog, применяемый в качестве средства разработки интеллектуальных систем.

Методы предотвращения нежелательного зациклования в рекурсивных логических программах были разработаны независимо Смитом и др. [1434], а также Тамаки и Сато [1487]. Кроме того, последняя статья включала данные о методе запоминания, предназначенном для логических программ, который интенсивно разрабатывался в качестве метода **табуированного логического программирования** Дэвидом С. Уорреном. Свифт и Уоррен [1483] показали, как дополнить машину WAM для обеспечения табуляции, что позволяет добиться быстродействия программ Datalog, превышающего на порядок быстродействие дедуктивных систем баз данных с прямым логическим выводом.

Первые теоретические работы по логическому программированию в ограничениях были выполнены Джффаром и Лассе [722]. Джффар и др. [723] разработали систему CLP(R) для обработки ограничений с действительными значениями. В [724] приведено описание обобщенной машины WAM, на основе которой создана машина CLAM (Constraint Logic Abstract Machine — абстрактная машина логики ограничений) для разработки спецификаций различных реализаций систем CLP. В [10] описан сложный язык Life, в котором методы CLP сочетаются с функциональным программированием и формированием рассуждений в логике наследования. В [820] описан перспективный проект использования логического программирования в ограничениях в качестве основы архитектуры управления в реальном времени, которая может применяться для создания полностью автоматических средств вождения самолетов (автопилотов).

Объем литературы по логическому программированию и языку Prolog очень велик. Одной из первых книг по логическому программированию явилась книга *Logic for Problem Solving* [851]. Языку Prolog посвящены, в частности, книги [175], [270] и [1405]. Превосходный обзор тематики CLP приведен в [987]. До его закрытия в 2000 году официальным журналом для публикаций в этой области был *Journal of Logic Programming*; теперь вместо него выпускается журнал *Theory and Practice of Logic Programming*. К числу основных конференций по логическому программированию

относятся *International Conference on Logic Programming* (ICLP) и *International Logic Programming Symposium* (ILPS).

Исследования в области **автоматического доказательства математических теорем** начались еще до того, как были впервые разработаны полные логические системы первого порядка. В разработанной Гербертом Гелернгером программе Geometry Theorem Prover [532] использовались методы эвристического поиска в сочетании с диаграммами для отсечения ложных подцелей; с помощью этой программы удалось обосновать некоторые весьма сложные результаты математических исследований в области евклидовой геометрии. Но с тех пор взаимодействие таких научных направлений, как автоматическое доказательство теорем и искусственный интеллект, не слишком велико.

На первых порах основные усилия ученых были сосредоточены на проблемах полноты. Вслед за появлением оригинальной статьи Робинсона в работах [1619] и [1620] были предложены правила демодуляции и парамодуляции для формирования рассуждений с учетом отношения равенства. Эти правила были также разработаны независимо в контексте систем перезаписи термов [811]. Внедрение средств формирования рассуждений с учетом отношения равенства в алгоритм унификации было осуществлено Гордоном Плоткиным [1217]; применение таких средств было также предусмотрено в языке Qlisp [1339]. В [752] приведен обзор средств унификации с учетом отношения равенства на основе процедур перезаписи термов. Эффективные алгоритмы для стандартной унификации были разработаны Мартелли и Монтанари [989], а также Патерсоном и Вегманом [1181].

Кроме средств формирования рассуждений с учетом отношения равенства, в программы автоматического доказательства теорем были включены всевозможные процедуры принятия решений специального назначения. В [1120] предложена получившая широкое распространение схема интеграции подобных процедур в общую схему формирования рассуждений; к другим методам относятся “результатия теории” Стикеля [1462] и “специальные отношения” Манна и Валдингера [978].

Для метода результата был предложен целый ряд стратегий управления, начиная со стратегии предпочтения единичного выражения [1616]. В [1617] была предложена стратегия с использованием множества поддержки, которая позволяет обеспечить определенную целенаправленность результата. Линейная результата впервые была предложена в [948]. В [537, глава 5] приведен краткий, но исчерпывающий анализ всего разнообразия стратегий управления.

В [602] описана одна из первых программ автоматического доказательства теорем, Sam, которая позволила решить одну из открытых проблем в теории решеток. В [1621] приведен краткий обзор того, какой вклад был внесен с помощью программы автоматического доказательства теорем Aura в решение открытых проблем в различных областях математики и логики. Это описание продолжено в [1018], где перечислены достижения программы Otter, преемника программы Aura, в решении открытых проблем. В [1563] описана программа Spass — одна из сильнейших современных программ автоматического доказательства теорем. Основным справочником по программе автоматического доказательства теорем Бойера–Мура является книга *A Computational Logic* [165]. В [1463] рассматривается система PTTP (Prolog Technology Theorem Prover), в которой сочетаются преимущества компиляции Prolog с полнотой устранения моделей [947]. Еще одной широко применяемой программой автоматического доказательства теорем, основанной на этом подходе, является SETHEO [915]; она способна выполнять несколько миллионов логических выводов

в секунду на рабочих станциях модели 2000. Эффективной программой автоматического доказательства теорем, реализованной всего лишь в 25 строках на языке Prolog, является LeanTaP [91].

Одни из первых работ в области автоматизированного синтеза программ были выполнены Саймоном [1416], Грином [591], а также Манна и Валдингером [976]. В трансформационной системе Бурстолла и Дарлингтона [211] используется формирование рассуждений с учетом отношения равенства для синтеза рекурсивных программ. Одной из самых сильных современных систем является Kids [1435], [1436]; она действует в качестве помощника эксперта. В [979] приведено учебное введение с описанием современного состояния дел в этой области, в котором основное внимание уделено описанию собственного дедуктивного подхода этих авторов. В книге *Automating Software Design* [954] собрано множество статей из этой области. Обзор примеров использования логики в проектировании аппаратных средств приведен в [791]; в [267] рассматривается применение метода проверки по модели для диагностирования аппаратных средств.

Хорошим справочником по темам полноты и неразрешимости является книга *Computability and Logic* [150]. Многие ранние статьи в области математической логики можно найти в книге *From Frege to Gödel: A Source Book in Mathematical Logic* [1532]. Официальным журналом для публикаций в области чистой математической логики (в отличие от автоматизированного дедуктивного логического вывода) является *The Journal of Symbolic Logic*. К числу учебников, посвященных автоматизированному дедуктивному логическому выводу, относятся классическая книга *Symbolic Logic and Mechanical Theorem Proving* [233], а также более новые работы [124], [776] и [1618]. Антология *Automation of Reasoning* [1408] включает много важных ранних статей по автоматизированному дедуктивному логическому выводу. Другие исторические обзоры приведены в [206] и [949]. Основным журналом для публикаций в области автоматического доказательства теорем является *Journal of Automated Reasoning*, а главной конференцией — ежегодно проводимая конференция *Conference on Automated Deduction* (CADE). Кроме того, исследования в области автоматического доказательства теорем тесно связаны с работами по использованию логики при анализе программ и языков программирования, которым посвящена основная конференция *Logic in Computer Science*.

## УПРАЖНЕНИЯ

- 9.1. Докажите на основании главных логических принципов, что процедура конкретизации с помощью квантора всеобщности является непротиворечивой и что процедура конкретизации с помощью квантора существования позволяет получить базу знаний, эквивалентную с точки зрения логического вывода.
- 9.2. Представляется вполне обоснованным утверждение, что из отдельного факта *Likes(Jerry, IceCream)* можно вывести высказывание  $\exists x \text{ Likes}(x, \text{IceCream})$ . Запишите общее правило логического вывода, правило **введения квантора существования**, позволяющее узаконить такой логический вывод. Тщательно сформулируйте условия, которым должны удовлетворять переменные и термы, участвующие в этом выводе.

- 9.3.** Предположим, что база знаний содержит только одно высказывание,  $\exists x \text{AsHighAs}(x, \text{Everest})$ . Какие из следующих фактов являются действительными результатами применения правила конкретизации с помощью квантора существования?
- $\text{AsHighAs}(\text{Everest}, \text{Everest})$ .
  - $\text{AsHighAs}(\text{Kilimanjaro}, \text{Everest})$ .
  - $\text{AsHighAs}(\text{Kilimanjaro}, \text{Everest}) \wedge \text{AsHighAs}(\text{BenNevis}, \text{Everest})$  (после двух применений).
- 9.4.** Для каждой приведенной ниже пары атомарных высказываний укажите наиболее общий унификатор, если он существует.
- $P(A, B, B), P(x, y, z)$ .
  - $Q(y, G(A, B)), Q(G(x, x), y)$ .
  - $\text{Older}(\text{Father}(y), y), \text{Older}(\text{Father}(x), \text{John})$ .
  - $\text{Knows}(\text{Father}(y), y), \text{Knows}(x, x)$ .
- 9.5.** Рассмотрите решетки обобщения, приведенные на рис. 9.1.
- Составьте решетку для высказывания  $\text{Employs}(\text{Mother}(\text{John}), \text{Father}(\text{Richard}))$ .
  - Составьте решетку для высказывания  $\text{Employs}(\text{IBM}, y)$  (“Компания IBM является нанимателем для всех”). Не забудьте включить запрос любого рода, который унифицируется с этим высказыванием.
  - Предположим, что функция *Store* индексирует каждое высказывание под каждым узлом в его решетке обобщения. Объясните, как должна работать функция *Fetch*, если некоторые из этих высказываний содержат переменные; воспользуйтесь в качестве примера высказываниями, приведенными в упр. 9.5, а и 9.5, б, а также запросом  $\text{Employs}(x, \text{Father}(x))$ .
- 9.6.** Предположим, что в логическую базу данных помещена часть данных переписи населения США с указанием возраста, города проживания, даты рождения и имени матери каждого лица, с использованием номеров карточек социального страхования в качестве идентифицирующих констант для каждого лица. Таким образом, например, возраст Джорджа задается выражением  $\text{Age}(443-65-1282, 56)$ . Какая из приведенных ниже схем индексации S1–S5 позволяет эффективно находить ответы на каждый из запросов Q1–Q4 (при условии, что применяется обычный метод обратного логического вывода)?
- Схемы индексации.
    - S1. Индекс для каждого атомарного терма в каждой позиции.
    - S2. Индекс для каждого первого параметра.
    - S3. Индекс для каждого атомарного терма предиката.
    - S4. Индекс для каждой комбинации предиката и первого параметра.
    - S5. Индекс для каждой комбинации предиката и второго параметра и индекс для каждого первого (нестандартного) параметра.
  - Запросы.
    - Q1.  $\text{Age}(443-44-4321, x)$ .
    - Q2.  $\text{ResidesIn}(x, \text{Houston})$ .

- Q3.  $Mother(x, y)$ .
  - Q4.  $Age(x, 34) \wedge ResidesIn(x, TinyTownUSA)$ .
- 9.7.** Можно было бы предположить, что стандартизация раз и навсегда отличий всех высказываний в базе знаний позволяет избежать проблемы конфликта переменных при унификации в процессе обратного логического вывода. Покажите, что для некоторых высказываний этот подход не применим. (*Подсказка*. Рассмотрите высказывание, одна часть которого унифицируется с другой.)
- 9.8.** Объясните, как записать любую конкретную формулировку задачи 3-SAT произвольного размера с использованием единственного определенного выражения первого порядка и не больше 30 базовых фактов.
- 9.9.** Запишите логические представления для приведенных ниже высказываний, применимые для использования с обобщенным правилом отделения.
- Лошади, коровы и свиньи — млекопитающие.
  - Рожденный лошадью — лошадь.
  - Блюбрёд — лошадь.
  - Блюбрёд — родитель Чарли.
  - Отношения “быть рожденным” и “быть родителем” — обратные.
  - Каждое млекопитающее имеет родителя.
- 9.10.** В этом упражнении для получения ответов на вопросы с помощью алгоритма обратного логического вывода используются высказывания, записанные при решении упр. 9.9.
- Нарисуйте дерево доказательства, сформированное исчерпывающим алгоритмом обратного логического вывода для запроса  $\exists h Horse(h)$  (“Существует некоторая лошадь”), в котором выражения согласуются в указанном порядке.
  - Какие особенности этой проблемной области вы обнаружили?
  - Какое количество решений для  $h$  фактически следует из ваших высказываний?
  - Можете ли вы предложить способ поиска всех этих решений? (*Подсказка*. Вам может потребоваться обратиться к [1434].)
- 9.11.** Одной из известных детских английских загадок является следующая: “Brothers and sisters have I none, but that man's father is my father's son” (Братьев и сестер у меня нет, но отец этого человека — сын моего отца). С использованием правил из области семейных отношений (глава 8) определите, кто этот человек, о котором говорится в загадке. Вы можете применять любые методы логического вывода, описанные в этой главе. Почему, по вашему мнению, эту загадку трудно отгадать сразу?
- 9.12.** Проследите за выполнением алгоритма обратного логического вывода, приведенного в листинге 9.3, при его применении для решения задачи доказательства преступления. Покажите, какую последовательность значений принимает переменная  $goals$ , и расположите эти значения в виде дерева.
- 9.13.** Приведенный ниже код Prolog определяет предикат  $P$ :
- ```
P(X, [X|Y]).  
P(X, [Y|Z]) :- P(X, Z).
```

- a) Покажите деревья доказательства и решения для запросов  $P(A, [1, 2, 3])$  и  $P(2, [1, A, 3])$ .
- б) Какую стандартную операцию со списками представляет предикат  $P$ ?
- 9.14.** В этом упражнении рассматривается применение сортировки в языке Prolog.
- a) Напишите выражения Prolog, которые определяют предикат  $\text{sorted}(L)$ , принимающий истинное значение тогда и только тогда, когда список  $L$  отсортирован в возрастающем порядке.
- б) Напишите на языке Prolog определение предиката  $\text{perm}(L, M)$ , который принимает истинное значение тогда и только тогда, когда  $L$  — перестановка  $M$ .
- в) Определите предикат  $\text{sort}(L, M)$  ( $M$  — отсортированная версия  $L$ ) с использованием предикатов  $\text{perm}$  и  $\text{sorted}$ .
- г) Применяйте предикат  $\text{sort}$  ко всем более длинным и длинным спискам, пока вам это не надоест. Какова временная сложность вашей программы?
- д) Реализуйте на языке Prolog более быстрый алгоритм сортировки, такой как сортировка вставкой (insert sort) или быстрая сортировка (quicksort).
- 9.15.** В этом упражнении рассматривается рекурсивное применение правил перезаписи с использованием логического программирования. Правилом перезаписи (или демодулятором, в терминологии программы Otter) является уравнение с указанным направлением применения. Например, правило перезаписи  $x+0 \rightarrow x$  указывает, что любое выражение, которое согласуется с  $x+0$ , должно заменяться выражением  $x$ . Средства применения правил перезаписи составляют центральную часть систем формирования рассуждений с учетом отношения равенства. Для представления правил перезаписи мы будем использовать предикат  $\text{rewrite}(X, Y)$ . Например, приведенное выше правило перезаписи может быть представлено как  $\text{rewrite}(X+0, X)$ . Некоторые термы являются примитивными и не могут подвергаться дальнейшим упрощениям, поэтому мы будем использовать запись  $\text{primitive}(0)$  для указания на то, что  $0$  — примитивный терм.
- а) Запишите определение предиката  $\text{simplify}(X, Y)$ , который принимает истинное значение, если  $Y$  — упрощенная версия  $X$ , т.е. к каким-либо подвыражениям  $Y$  больше не применимы какие-либо правила перезаписи.
- б) Запишите коллекцию правил перезаписи для упрощения выражений, в которых применяются арифметические операторы, и примените ваш алгоритм упрощения к некоторым примерам выражений.
- в) Запишите коллекцию правил перезаписи для символьского дифференцирования и примените их наряду с определенными вами правилами упрощения для дифференцирования и упрощения выражений, в которых есть арифметические выражения, включая возвведение в степень.
- 9.16.** В этом упражнении рассматривается реализация алгоритмов поиска на языке Prolog. Предположим, что предикат  $\text{successor}(X, Y)$  принимает истинное значение, если состояние  $Y$  является преемником состояния  $X$ , и что предикат  $\text{goal}(X)$  принимает истинное значение, если  $X$  — целевое состояние. Запишите определение для предиката  $\text{solve}(X, P)$ , который означает, что  $P$  — путь (список состояний), начинающийся от  $X$ , оканчивающийся в целевом

состоянии и состоящий из последовательности допустимых шагов, которые определены предикатом `successor`. Вы обнаружите, что простейшим способом решения этой задачи является поиск в глубину. Насколько легко будет ввести эвристическое управление поиском?

- 9.17.** Как можно воспользоваться методом резолюции для демонстрации того, что некоторое высказывание является общезначимым? Невыполнимым?
- 9.18.** Из высказывания “Лошади — животные” следует, что “голова лошади — голова животного”. Продемонстрируйте, что этот логический вывод является допустимым, выполнив приведенные ниже этапы.
- Преобразуйте предпосылку и вывод этого высказывания в язык логики первого порядка. Воспользуйтесь тремя предикатами: `HeadOf(h, x)` (который означает, что “ $h$  — голова  $x$ ”), `Horse(x)` и `Animal(x)`.
  - Примените отрицание к заключению и преобразуйте предпосылку и отрицаемое заключение в конъюнктивную нормальную форму.
  - Воспользуйтесь правилом резолюции, чтобы показать, что заключение следует из предпосылки.
- 9.19.** Ниже приведены два высказывания на языке логики первого порядка.
- $\forall x \exists y (x \geq y)$
  - $\exists y \forall x (x \geq y)$
- Допустим, что переменные пробегают по всем натуральным числам  $0, 1, 2, \dots, \infty$  и что предикат  $\geq$  означает “больше или равно”. При использовании такой интерпретации переведите высказывания *A* и *B* на естественный язык.
  - Является ли высказывание *A* истинным при этой интерпретации?
  - Является ли высказывание *B* истинным при этой интерпретации?
  - Является ли *B* логическим следствием *A*?
  - Является ли *A* логическим следствием *B*?
  - С использованием правила резолюции попытайтесь доказать, что *A* следует из *B*. Сделайте эту попытку, даже если вы считаете, что *A* не следует логически из *B*; продолжайте свои усилия до тех пор, пока доказательство не оборвётся и вы не сможете продолжать дальше (поскольку оно оборвалось). Покажите унифицирующую подстановку для каждого этапа резолюции. Если доказательство окончилось неудачей, точно объясните, где, как и почему оно оборвалось.
  - А теперь попытайтесь доказать, что *B* следует из *A*.
- 9.20.** Метод резолюции способен вырабатывать неконструктивные доказательства для запросов с переменными, поэтому приходится вводить специальные механизмы для извлечения только определенных ответов. Объясните, почему такая проблема не возникает при использовании баз знаний, содержащих только определенные выражения.
- 9.21.** В этой главе было указано, что метод резолюции не может использоваться для формирования всех логических следствий из некоторого множества высказываний. Позволяет ли какой-то другой алгоритм решить эту задачу?

# 10 ПРЕДСТАВЛЕНИЕ ЗНАНИЙ

*В этой главе показано, как использовать логику первого порядка для представления наиболее важных аспектов реального мира, таких как действия, пространство, время, мыслимые события, а в качестве примера описано осуществление покупок.*

В предыдущих трех главах была описана технология агентов, основанных на знаниях: синтаксис, семантика и теория доказательства для пропозициональной логики и логики первого порядка, а также реализация агентов, в которых применяются эти варианты логики. В данной главе рассматривается вопрос о том, какое информационное наполнение следует поместить в базу знаний такого агента, т.е. как представлять факты о мире.

В разделе 10.1 описывается идея общей онтологии, которая позволяет организовать все, что существует в мире, в виде иерархии категорий. В разделе 10.2 рассматриваются основные категории объектов, веществ и мер. В разделе 10.3 обсуждаются способы представления для действий, которые играют центральную роль в создании агентов, основанных на знаниях, а также раскрывается более общее понятие **событий**, или пространственно-временных фрагментов. В разделе 10.4 излагаются знания об убеждениях, а в разделе 10.5 все эти знания переводятся в контекст среды осуществления покупок в Internet. В разделах 10.6 и 10.7 рассматриваются специализированные системы формирования рассуждений для представления неопределенных и изменяющихся знаний.

## 10.1. ОНТОЛОГИЧЕСКАЯ ИНЖЕНЕРИЯ

В “учебных”, экспериментальных проблемных областях выбор представления не столь важен; для них несложно найти подходящий словарь. С другой стороны, для таких сложных проблемных областей, как осуществление покупок в Internet или управление роботом в изменяющейся физической среде, требуются более общие и гибкие способы представления. В настоящей главе показано, как создать такие представления, сосредоточиваясь на общих понятиях (таких как действия, время, физические объекты и убеждения), которые проявляются во многих разных про-

блемных областях. Способы представления таких абстрактных понятий иногда называют **онтологической инженерией**; процесс применения этих способов связан с процессом **инженерии знаний**, описанным в разделе 8.4, но разворачивается в больших масштабах.

Попытка формально представить все, что существует в мире, была бы бесперспективной. Безусловно, нам не требуется фактически составлять полное описание всего, что существует в мире (требуемый для этого объем изложения был бы слишком велик даже для учебника с 1000 страниц), но мы можем оставить свободные места, в которые будут укладываться новые знания о любой проблемной области. Например, мы определим, что подразумевается под понятием *физический объект*, а описания подробных сведений о различных типах объектов (роботах, телевизорах, книгах или о чем-то другом) оставим на будущее. Такая общая инфраструктура понятий называется **верхней онтологией**, поскольку принято соглашение составлять схемы онтологии, помещая общие понятия в верхней части, а более конкретные понятия — под ними, как показано на рис. 10.1.

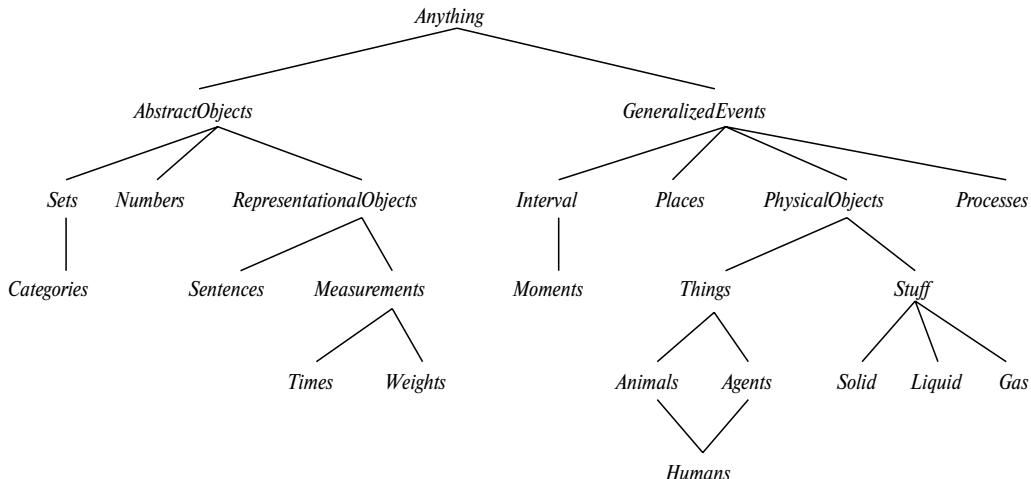


Рис. 10.1. Всемирная онтология, в которой показаны темы, рассматриваемые ниже в данной главе. Каждая дуга указывает на нижнее понятие, которое является уточнением верхнего

Прежде чем приступить к более подробному обсуждению онтологии, необходимо высказать одно важное предостережение. Мы решили использовать логику первого порядка для обсуждения содержания и организации знаний. Но некоторые аспекты реального мира трудно представить в логике первого порядка. Основная сложность состоит в том, что почти все обобщения имеют исключения или соблюдаются только до определенной степени. Например, хотя правило “помидоры имеют красный цвет” является достаточно удобным, некоторые помидоры имеют зеленый, желтый или оранжевый цвет. Аналогичные исключения можно указать применительно почти ко всем общим утверждениям из этой главы. Способность учитывать исключения и неопределенность является чрезвычайно важной, но она ортогональна задаче понимания общей онтологии. По этой причине мы отложим обсуждение исключений до раздела 10.6, а более общей темы неопределенной информации — до главы 13.

Для чего может применяться верхняя онтология? Еще раз рассмотрим онтологию электронных схем, описанную в разделе 8.4. В ней принято большое количество упрощающих допущений. Например, полностью исключено понятие времени. Сигналы являются постоянными и не распространяются. Структура схемы остается неизменной. А если бы потребовалось сделать эту онтологию более общей, то нужно было бы учитывать, какие значения сигналы имеют в конкретные моменты времени, а также вводить данные о длине проводников и задержках распространения. Это позволило бы моделировать временные свойства схемы, и такое моделирование действительно часто осуществляется проектировщиками схем. Кроме того, можно было бы ввести более широкий перечень интересующих классов электронных элементов, например, описывая технологию (ТТЛ, МОП, КМОП и т.д.), а также спецификацию ввода-вывода. Если же нам потребовалось бы обсуждать надежность или проводить диагностику, то необходимо было бы учесть возможность неконтролируемого изменения структуры схемы или свойств электронных элементов. Чтобы учесть паразитные емкости, нужно было бы перейти от чисто топологического представления связей к более реалистичному описанию геометрических свойств.

Рассматривая мир вампуса, можно прийти к аналогичным выводам. Хотя в описание этого мира включено время, оно имеет очень простую структуру: в мире ничего не происходит, кроме действий агента, а все изменения осуществляются мгновенно. Более общая онтология, которая лучше подходила бы для описания реального мира, должна была бы позволить синхронным изменениям распространяться во времени. Кроме того, для представления информации о том, в каких квадратах имеются ямы, использовался предикат *Pit*. Но можно было бы разрешить применять ямы разных типов, определив несколько объектов, принадлежащих к конкретному классу ям, каждый из которых характеризуется разными свойствами. Кроме того, в этом мире могла быть предусмотрена возможность обеспечить существование других животных, кроме вампусов. В таком случае не было бы возможности точно определять вид встретившегося животного из доступных результатов восприятия, поэтому потребовалось бы разработать биологическую таксономию для мира вампуса, позволяющую агенту прогнозировать оптимальное поведение на основании скучой информации.

При использовании любой онтологии специального назначения существует возможность вносить изменения, подобные этим, для перехода к большей общности. В таком случае возникает очевидный вопрос — сходятся ли все эти онтологии к некоторой онтологии общего назначения? По истечении целых столетий философских и математических исследований на этот вопрос был дан лишь один ответ — “Возможно”. В данном разделе представлена одна из таких версий, воплощающая в себя синтез идей, собранных в течение многих веков. Онтологии общего назначения обладают двумя описанными ниже основными особенностями, которые отличают их от коллекций отдельных онтологий специального назначения.

- Любая онтология общего назначения должна быть в большей или меньшей степени применимой для любой специализированной проблемной области (с добавлением аксиом конкретной проблемной области). Это означает, что в ней по мере возможности не следует ни детализировать, ни скрывать какие-либо вопросы конкретного представления.

- В любой проблемной области, характеризующейся достаточно высокой значимостью, различные области знаний должны быть унифицированы, поскольку процессы формирования рассуждений и решения задач могут включать сразу несколько направлений одновременно. Например, в системе ремонта электронных схем робота необходимо формировать рассуждения и о самих схемах, с точки зрения состояния электрических соединений и физической компоновки, и о времени, поскольку последнее требуется для анализа временных свойств схем и оценки необходимых затрат труда на ремонт. Поэтому высказывания с описанием времени должны быть применимыми для комбинирования с высказываниями, описывающими пространственную компоновку, и одинаково хорошо представлять не только наносекунды и минуты, но и ангстремы и метры.

После представления в этой главе общей онтологии мы применим ее для описания проблемной области осуществления покупок в Internet. Эта проблемная область чрезвычайно хорошо подходит для упражнений в области онтологии, а также оставляет для читателя большой простор, чтобы он мог сам создавать творческие представления знаний. Достаточно представить себе, например, что агент для осуществления покупок в Internet должен знать о бесчисленном множестве тем и авторов, чтобы покупать книги на узле Amazon.com, владеть информацией о всевозможных сортах пищевых продуктов, чтобы покупать бакалейные товары на узле Reardon.com, а также иметь сведения обо всем, что можно найти на дешевой распродаже, чтобы не упустить выгодные сделки на узле Ebay.com<sup>1</sup>.

## 10.2. КАТЕГОРИИ И ОБЪЕКТЫ

Крайне важной частью любого способа представления знаний является классификация объектов по **категориям**. Хотя взаимодействие с миром происходит на уровне отдельных объектов, *формирование рассуждений в основном происходит на уровне категорий*. Например, покупатель может иметь перед собой цель купить хотя бы какой-то баскетбольный мяч, а не конкретный баскетбольный мяч, допустим, обозначенный номером BB<sub>9</sub>. Кроме того, категории позволяют многое предсказывать в отношении объектов после того, как эти объекты вошли в состав классификации. При этом делается заключение о наличии некоторых объектов на основании сенсорных входных данных, определяется принадлежность к категории из воспринятых свойств объектов, а затем информация о категории используется для составления прогнозов, касающихся этих объектов. Например, на основании того, что некоторый объект имеет зеленый цвет, полосатую поверхность, большой размер и круглую форму, можно сделать вывод, что это — арбуз, а исходя из этого заключить, что данный объект подходит для использования в качестве сладкого десерта.

Для представления категорий в логике первого порядка могут применяться два основных способа: представление с помощью предикатов или с помощью объектов. Это означает, что для этого можно либо применить предикат, такой как *Basketball(b)*,

<sup>1</sup> Авторы заранее приносят извинения, если по независящим от них обстоятельствам некоторые из упомянутых здесь оперативных магазинов уже не будут функционировать к тому времени, как эта книга попадет в руки читателя.

либо **овеществить** всю категорию баскетбольных мячей в виде некоторого объекта — множества баскетбольных мячей *Basketballs*. После этого можно сформировать высказывание *Member*(*b*, *Basketballs*) (которое мы будем сокращенно записывать как *b* ∈ *Basketballs*) в качестве утверждения, что *b* — элемент категории баскетбольных мячей. Высказывание *Subset*(*Basketballs*, *Balls*) (сокращенно обозначаемое как *Basketballs* ⊂ *Balls*) может применяться в качестве утверждения, что *Basketballs* — подкатегория, или подмножество мячей *Balls*. Поэтому любую категорию можно рассматривать как множество элементов или считать ее более сложным объектом и полагать, будто просто так оказалось, что для этого объекта определены отношения *Member* и *Subset*.

Категории служат для организации и упрощения базы знаний с помощью **наследования**. Если известно, что все экземпляры категории *Food* (пища) съедобны, и сформулировано утверждение, что *Fruit* (фрукты) — это подкласс класса *Food*, а *Apples* (яблоки) — подкласс класса *Fruit*, то становится известно, что каждое яблоко съедобно. Это определение формулируется таким образом, что отдельные яблоки **наследуют** свойства съедобности, в данном случае в силу своей принадлежности к категории *Food*.

Отношения между классами и подклассами позволяют организовывать категории в виде некоторой **таксономии**, или **таксономической иерархии**. Явно заданные таксономии использовались в прикладных науках в течение многих столетий. Например, предметом систематической биологии является создание таксономии для всех существующих и исчезнувших видов; в библиографии разработана таксономия всех областей знаний, закодированная в виде десятичной системы Дьюи; а налоговые органы и другие правительственные организации разработали обширные таксономии профессий и коммерческих товаров. Кроме того, таксономия представляет собой важный аспект общих повседневных знаний.

С другой стороны, логика первого порядка позволяет легко формулировать факты о категориях, либо связывая объекты с категориями, либо применяя кванторы к их элементам, как описано ниже.

- Любой объект — элемент некоторой категории, например:  
 $BB_9 \in Basketballs$
- Любая категория — подкласс другой категории, например:  
 $Basketballs \subset Balls$
- Все элементы категории имеют некоторые свойства, например:  
 $x \in Basketballs \Rightarrow Round(x)$
- Элементы категории могут быть распознаны по некоторым свойствам, например:  
 $Orange(x) \wedge Round(x) \wedge Diameter(x) = "9.5" \wedge x \in Balls \Rightarrow x \in Basketballs$
- Вся категория в целом имеет некоторые свойства, например:  
 $Dogs \in DomesticatedSpecies$

Обратите внимание на то, что *Dogs* (Собаки) — и категория, и элемент категории *DomesticatedSpecies* (Домашние животные), поэтому последняя должна

быть категорией категорий. Могут даже существовать категории категории категорий, но область их применения не так уж велика.

Хотя отношения между подклассами и классами, а также между элементами и множествами являются для категорий наиболее важными, необходимо также иметь возможность формулировать отношения между категориями, которые не являются подклассами друг друга. Например, если будет отмечено, что *Males* (Самцы) и *Females* (Самки) — подклассы класса *Animals* (Животные), то этим не будет сказано, что ни один самец не может одновременно быть самкой. Две или несколько категорий являются **непересекающимися**, если они не имеют общих элементов. И даже если известно, что категории самцов и самок не пересекаются, на этом основании нельзя утверждать, что животное, не являющееся самцом, должно быть самкой, если не сформулировано дополнительное утверждение, что самцы и самки образуют **исчерпывающую декомпозицию** категории животных. Непересекающуюся исчерпывающую декомпозицию принято называть **сегментацией**. Эти три понятия иллюстрируются в приведенных ниже примерах.

```
Disjoint({Animals, Vegetables})
ExhaustiveDecomposition({Americans, Canadians, Mexicans},
NorthAmericans)
Partition({Males, Females}, Animals)
```

(Обратите внимание на то, что исчерпывающая декомпозиция *ExhaustiveDecomposition* североамериканцев *NorthAmericans* не является сегментацией *Partition*, поскольку некоторые люди имеют двойное гражданство.) Эти три предиката определены следующим образом:

```
Disjoint(s)  $\Leftrightarrow (\forall c_1, c_2 \ c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow \text{Intersection}(c_1, c_2) = \{\})$ 
ExhaustiveDecomposition(s, c)  $\Leftrightarrow (\forall i \ i \in c \Leftrightarrow \exists c_2 \ c_2 \in s \wedge i \in c_2)$ 
Partition(s, c)  $\Leftrightarrow \text{Disjoint}(s) \wedge \text{ExhaustiveDecomposition}(s, c)$ 
```

Категории могут быть также определены путем указания необходимых и достаточных условий принадлежности к ним. Например, холостяк — это неженатый взрослый мужчина:

```
x  $\in$  Bachelors  $\Leftrightarrow \text{Unmarried}(x) \wedge x \in \text{Adults} \wedge x \in \text{Males}$ 
```

Как описано во врезке “Естественные разновидности”, строгие логические определения для категорий либо не всегда возможны, либо не всегда необходимы.

## Физическая композиция

Идея о том, что один объект может составлять часть другого, для нас весьма привычна. Нос — это часть лица, Румыния — часть Европы, а данная глава — часть настоящей книги. Для указания на то, что одна вещь является частью другой, используется общее отношение *PartOf*. Объекты могут группироваться в иерархии *PartOf*, напоминающие иерархию *Subset* (подмножество), например, как показано ниже.

```
PartOf(Bucharest, Romania)
PartOf(Romania, EasternEurope)
PartOf(EasternEurope, Europe)
PartOf(Europe, Earth)
```

Отношение *PartOf* является транзитивным и рефлексивным, т.е. для него справедливы следующие высказывания:

$$\begin{aligned} \textit{PartOf}(x, y) \wedge \textit{PartOf}(y, z) &\Rightarrow \textit{PartOf}(x, z) \\ \textit{PartOf}(x, x) \end{aligned}$$

Поэтому можно сделать вывод, что *PartOf(Bucharest, Earth)*.

Категории ~~составных объектов~~ часто характеризуются структурными отношениями между частями. Например, любое двуногое имеет две и только две ноги, прикрепленные к телу:

$$\begin{aligned} \textit{Biped}(a) \Rightarrow \exists l_1, l_2, b \textit{Leg}(l_1) \wedge \textit{Leg}(l_2) \wedge \textit{Body}(b) \wedge \\ \textit{PartOf}(l_1, a) \wedge \textit{PartOf}(l_2, a) \wedge \textit{PartOf}(b, a) \wedge \\ \textit{Attached}(l_1, b) \wedge \textit{Attached}(l_2, b) \wedge \\ l_1 \neq l_2 \wedge [\forall l_3 \textit{Leg}(l_3) \wedge \textit{PartOf}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)] \end{aligned}$$

Применяемая здесь система обозначения понятия “две и только две” является довольно громоздкой; мы были вынуждены указать, что ног две, что ноги не являются одинаковыми и что если кто-то из двуногих будет утверждать, что у него есть третья нога, она в конечном итоге окажется одной из его двух ног. В разделе 10.6 будет показано, что формальная система, называемая *описательной логикой*, позволяет проще представить ограничения типа “две и только две”.

Для категорий может быть определено отношение *PartPartition*, аналогичное отношению *Partition* (см. упр. 10.6). Любой объект состоит из частей, принадлежащих к его категориям *PartPartition*, и может рассматриваться как получающий некоторые свойства от собственных частей. Например, масса составного объекта — это сумма масс его частей. Обратите внимание на то, что это утверждение не распространяется на категории, не имеющие массы, даже если эти категории состоят из элементов, обладающих массой.

Кроме того, целесообразно определить составные объекты, имеющие различные части, но не имеющие конкретной структуры. Например, может потребоваться сформулировать утверждение: “Яблоки в этой сумке весят два килограмма”. Может возникнуть соблазн приписать этот вес множеству яблок в сумке, но это было бы ошибкой, поскольку множество — это абстрактное математическое понятие, которое имеет элементы, но не имеет веса. Вместо этого необходимо ввести новое понятие, которое мы будем именовать ~~совокупностью~~ (*BunchOf*). Например, если три яблока обозначены как *Apple<sub>1</sub>*, *Apple<sub>2</sub>* и *Apple<sub>3</sub>*, то выражение

$$\textit{BunchOf}(\{\textit{Apple}_1, \textit{Apple}_2, \textit{Apple}_3\})$$

обозначает составной объект, частями (а не элементами) которого являются три яблока. Затем эта совокупность может использоваться как обычный, хотя и не структурированный объект. Обратите внимание на то, что *BunchOf(\{x\}) = x*. Кроме того, *BunchOf(Apples)* — это составной объект, состоящий из всех яблок, но его не следует путать с объектом *Apples* — категорией или множеством всех яблок.

Определим понятие *BunchOf* в терминах отношения *PartOf*. Очевидно, что каждый элемент множества *s* — это часть объекта *BunchOf(s)*:

$$\forall x \ x \in s \Rightarrow \textit{PartOf}(x, \textit{BunchOf}(s))$$

Более того,  $BunchOf(s)$  — это наименьший объект, удовлетворяющий данному условию. Иными словами,  $BunchOf(s)$  должен быть частью любого объекта, который включает все элементы множества  $s$  в качестве части:

$$\forall y [\forall x x \in s \Rightarrow PartOf(x, y)] \Rightarrow PartOf(BunchOf(s), y)$$

Эти аксиомы представляют собой пример общего метода, называемого **логической минимизацией**, с помощью которого любой объект может быть определен как наименьший объект, удовлетворяющий определенным условиям.

### ЕСТЕСТВЕННЫЕ РАЗНОВИДНОСТИ

Некоторые категории имеют строгие определения, например, некоторый объект рассматривается как треугольник тогда и только тогда, когда он является многоугольником с тремя сторонами. С другой стороны, большинство категорий в реальном мире не имеют четких определений; они называются категориями **естественных разновидностей**. Например, помидоры, как правило, имеют красный цвет; их форма приближается к шарообразной; там, где в верхней части был черенок, остается углубление; диаметр их составляет примерно от пяти до десяти сантиметров; кожица тонкая, но прочная; внутри находятся мякоть, семена и сок. Тем не менее существуют экземпляры, не совсем подходящие под это описание: некоторые помидоры — оранжевые, незрелые помидоры — зеленые, бывают помидоры больше или меньше среднего, а все помидоры-сливки одинаково малы. Вместо наличия полного определения помидоров мы имеем дело с набором характеристик, позволяющих выявлять объекты, которые, безусловно, представляют собой типичные помидоры, но могут не позволить отличить их от других объектов. (Может быть, скоро появятся даже помидоры, покрытые пушком, как персики?)

В связи с этим перед любым логическим агентом возникает проблема. Агент не может быть уверен в том, что некоторый объект, ставший предметом его восприятия, действительно является помидором, и даже если бы он был уверен в этом, то не смог бы с уверенностью утверждать, какими свойствами типичных помидоров обладает данный объект. Эта проблема является неизбежным следствием того, что агенту приходится действовать в частично наблюдаемых вариантах среды.

Один из полезных подходов состоит в том, что нужно отделить сведения, истинные для всех экземпляров некоторой категории, от сведений, истинных только для типичных экземпляров. Поэтому, кроме категории *Tomatoes*, необходимо также предусмотреть категорию *Typical(Tomatoes)*. Здесь функция *Typical* отображает категорию на подкласс, который содержит только типичные экземпляры:

$$Typical(c) \subseteq c$$

Основной объем знаний об естественных разновидностях фактически относится к их типичным экземплярам:

$$x \in Typical(Tomatoes) \Rightarrow Red(x) \wedge Round(x)$$

Таким образом, мы получаем возможность формулировать полезные факты о категориях без точных определений.

Сложности подготовки точных определений для большинства естественных категорий были глубоко исследованы Виттгенштейном [1608] в его книге *Philosophical Investigations*. Он использовал пример с играми, чтобы показать, что элементы любой категории обладают, скорее, “семейным сходством”, а не необходимыми и достаточными для их классификации характеристиками.

Полезность понятия строгого определения была также поставлена под сомнение Квайном [1252], который указал, что небесспорно даже определение “холостяка” как неженатого взрослого мужчины. Например, вряд ли является корректным такое утверждение, что “Папа Римский — холостяк”. Хотя это утверждение, строго говоря, не является ложным, сам способ употребления в нем несовместимых слов, безусловно, не заслуживает одобрения, поскольку вызывает у слушателя нежелательные ассоциации. Возможно, указанную проблему можно было бы решить, проводя различия между логическими определениями, подходящими для внутреннего представления знаний, и более утонченными критериями, касающимися корректного лингвистического словоупотребления. Последней цели можно было бы достичь, “фильтруя” утверждения, полученные после достижения первой. Кроме того, возможно, что неудачные результаты при лингвистической проверке правильности словоупотребления могут использоваться в качестве обратной связи для модификации внутренних определений, чтобы такая фильтрация стала ненужной.

## Меры

И в научных, и в обыденных теориях мира объекты имеют высоту, массу, стоимость и т.д. Значения, применяемые для оценки этих свойств, называются мерами. Обычные количественные меры можно представить довольно легко. Предположим, что вселенная включает абстрактные “объекты мер”, такие как длина, которая представляет собой длину следующего отрезка прямой:  $\text{¶}^{\circ}\Phi\Omega\alpha!$   $\text{¶}\Omega\alpha\alpha\theta\epsilon\sigma\ \phi\sigma\Gamma\sigma=\theta\epsilon\sigma\ F\omega\square\ \text{¶}\alpha\theta\delta\alpha..$  Этот объект можно назвать отрезком с длиной 1,5 дюйма, или 3,81 сантиметра. Поэтому одна и та же длина может иметь в естественном языке разные имена. С точки зрения логики такую особенность можно учесть, комбинируя функцию единиц измерения с числом (альтернативная схема рассматривается в упр. 10.8). Если приведенному выше отрезку прямой присвоено имя  $L_1$ , то можно записать следующее:

$$\text{Length}(L_1) = \text{Inches}(1.5) = \text{Centimeters}(3.81)$$

Преобразование из одних единиц измерения в другие осуществляется путем прививания кратных значений одной единицы другой:

$$\text{Centimeters}(2.54 \times d) = \text{Inches}(d)$$

Аналогичные аксиомы могут быть записаны для фунтов и килограммов, секунд и суток, долларов и центов. Меры могут использоваться для описания объектов следующим образом:

$$\text{Diameter}(\text{Basketball}_{12}) = \text{Inches}(9.5)$$

$$\text{ListPrice}(\text{Basketball}_{12}) = \$ (19)$$

$$d \in \text{Days} \Rightarrow \text{Duration}(d) = \text{Hours}(24)$$

Обратите внимание на то, что  $\$ (1)$  — это не долларовая купюра! Некто может иметь две долларовые купюры, но существует только один объект с именем  $\$ (1)$ .

Следует также отметить, что объекты *Inches(0)* и *Centimeters(0)* обозначают одну и ту же нулевую длину, но не идентичны другим нулевым мерам, таким как *Seconds(0)*.

Простые, количественные меры можно представить легко, а проблема представления других мер становится более сложной, поскольку для них не предусмотрена согласованная шкала значений. Упражнения характеризуются сложностью, десерты — тонким вкусом, а стихи — красотой, но эти качества не могут быть обозначены числами. Можно было бы, стремясь к полной описуемости всего, отказаться от этих свойств как бесполезных с точки зрения логических рассуждений, или, что еще хуже, попытаться оценить красоту по числовой шкале. Но такое решение было бы грубой ошибкой, поскольку оно отнюдь не является необходимым. Наиболее важным свойством мер являются не конкретные числовые значения, а тот факт, что меры могут быть упорядочены.

Хотя многие меры не являются числами, все еще остается возможность сравнивать их с помощью некоторого символа упорядочения, такого как  $>$ . Например, можно предположить, что упражнения, составленные Норвигом, — более трудные, чем составленные Расселом, и что сложные упражнения не позволяют добиться высоких оценок, как показано ниже.

$$\begin{aligned} e_1 \in Exercises \wedge e_2 \in Exercises \wedge Wrote(Norvig, e_1) \wedge \\ Wrote(Russell, e_2) \Rightarrow \\ Difficulty(e_1) > Difficulty(e_2) \\ e_1 \in Exercises \wedge e_2 \in Exercises \wedge Difficulty(e_1) > Difficulty(e_2) \Rightarrow \\ ExpectedScore(e_1) < ExpectedScore(e_2) \end{aligned}$$

Этого достаточно, чтобы можно было принять решение о том, какие упражнения следует выбирать на экзаменах, даже несмотря на то, что для оценки их сложности не использовались какие-либо числовые значения (тем не менее для этого необходимо иметь возможность определить, кем составлены те или иные упражнения). Такого рода монотонные связи между мерами образуют основу для **качественной физики** — одного из направлений искусственного интеллекта, в котором изучаются способы формирования рассуждений о физических системах без погружения в числовые расчеты и подробные уравнения. Качественная физика обсуждается в разделе с историческими заметками.

## Вещества и объекты

Реальный мир может рассматриваться как составленный из примитивных объектов (частиц) и сложных объектов, состоящих из примитивных. Проведение рассуждений на уровне больших объектов, таких как яблоки и автомобили, позволяет преодолеть сложности, связанные с учетом бесконечных количеств примитивных объектов, отдельно взятых. Тем не менее существует значительная часть реальности, которая, по-видимому, не поддается никаким явным попыткам  $\bowtie$  индивидуализации — деления на отдельные объекты. Мы присвоим этой части мира универсальное имя —  $\bowtie$  **вещество**. Например, предположим, что перед исследователем находится немного масла и муравьед. Аналитик может утверждать, что перед ним один муравьед, но не в состоянии указать какое-то явное количество “масло-объектов”, поскольку любая часть масло-объекта — это также масло-объект, по крайней мере, до тех пор, пока не удастся дойти действительно до очень мелких его частей. В этом и состоит основное различие между веще-

ствами и объектами. Две половинки муравьеда, разрезанного пополам, уже нельзя, к сожалению, назвать двумя муравьедами.

Во многих языках, в том числе в английском, проводится четкое различие между веществами и объектами. Например, в нем можно употребить неопределенный artikel со словом “муравьед” — “an aardvark”, но нигде, кроме претенциозных калифорнийских ресторанов, не приходится слышать выражение “a butter”, обозначающее порцию масла. Лингвисты проводят различие между **исчисляемыми существительными**, такими как муравьеды, ямы и теоремы, и **неисчисляемыми существительными**, такими как масло, вода и энергия. Разработано несколько конкурирующих онтологий, которые претендуют на то, что в них учитывается это различие. В данной главе будет описана только одна из них, а остальные рассматриваются в разделе с историческими заметками.

Чтобы правильно представить понятие вещества, начнем с очевидных соображений. Нам необходимо иметь в своей онтологии в качестве объектов, по меньшей мере, достаточно большие “куски” вещества, чтобы описать способы взаимодействия с этими объектами. Например, в качестве куска масла можно было бы рассматривать то масло, которое осталось на столе от вчерашнего ужина; его можно собрать, взвесить, продать или проделать какие-то другие действия. В этом смысле кусок масла представляет собой объект, полностью аналогичный муравьеду. Назовем его *Butter<sub>3</sub>*. Кроме того, определим категорию *Butter*. Неформально всеми его элементами будут такие объекты, о которых можно сказать “Это — масло”, включая *Butter<sub>3</sub>*. Если не учитывать некоторые предостережения об очень малых частях вещества, которые мы пока не рассматриваем, то любая часть масла-объекта также представляет собой масло-объект:

$$x \in \text{Butter} \wedge \text{PartOf}(y, x) \Rightarrow y \in \text{Butter}$$

Теперь мы можем утверждать, что масло плавится при температуре около 30 градусов по Цельсию:

$$x \in \text{Butter} \Rightarrow \text{MeltingPoint}(x, \text{Centigrade}(30))$$

Продолжая его описание, можно сказать, что масло — желтое, имеет плотность меньше воды, размягчается при комнатной температуре, содержит много жиров и т.д. С другой стороны, масло не имеет конкретного размера, формы или веса. Могут быть определены более специализированные категории масла, такие как *UnsaltedButter* (Несоленое масло), которые также представляют собой разновидность вещества. С другой стороны, категория *PoundOfButter* (Фунт масла), которая включает в качестве элементов все масло-объекты, весящие один фунт, не является веществом! Если мы разрежем фунт масла пополам, то, к сожалению, не получим два фунта масла.

В этом анализе фактически обнаружилось следующее: существуют некоторые свойства, которые являются **внутренними**, — они принадлежат к каждой части объекта, а не к самому объекту в целом. Разрезав кусок вещества пополам, получим две части, обладающие одним и тем же набором внутренних свойств, таких как плотность, точка кипения, запах, цвет, принадлежность и т.д. С другой стороны, **внешние** свойства прямо противоположны — вес, длина, форма, назначение и пр. не сохраняются после разделения на части.

Поэтому тот класс объектов, который включает в свое определение только внутренние свойства, обозначается качественными, или неисчисляемыми существительными, а класс, который включает в свое определение только внешние свойства, обозначается количественными, или исчисляемыми существительными. Категория *Stuff* — это наиболее общая категория вещества, в которой не задаются какие-либо внутренние свойства, а категория *Thing* — это наиболее общая категория различных объектов, в которой не задаются какие-либо внешние свойства. Все физические объекты принадлежат к обеим категориям, поэтому эти категории коэкстенсивны (равнообъемны) — они относятся к одним и тем же сущностям.

### 10.3. ДЕЙСТВИЯ, СИТУАЦИИ И СОБЫТИЯ

Формирование рассуждений о результатах действий является центральным звеном в функционировании агента, основанного на знаниях. В главе 7 приведены примеры пропозициональных высказываний, описывающих, как действия влияют на мир вампуса, например, в уравнении 7.3 на с. 325 указано, как изменяется местонахождение агента в результате движения вперед. Одним из недостатков пропозициональной логики является необходимость иметь отдельную копию описания действия для каждого момента времени, в который может быть выполнено это действие. В данном разделе описан метод представления с использованием логики первого порядка, позволяющий избежать возникновения этой проблемы.

#### Онтология ситуационного исчисления

Один из очевидных способов предотвращения необходимости создавать многочисленные копии аксиом состоит в том, чтобы применить квантификацию по времени и использовать примерно такие высказывания: “ $\forall t$  (для всех  $t$ ) *то-то и то-то* является полученным в момент времени  $t+1$  результатом выполнения *такого-то* действия в момент времени  $t$ ”. Но в этом разделе вместо применения таких явных обозначений времени, как  $t+1$ , мы сосредоточимся на описании ситуаций, которые обозначают состояния, возникающие в результате выполнения действий. Такой подход, называемый **ситуационным исчислением**, предусматривает использование описанной ниже онтологии.

- Как и в главе 8, действия представляют собой логические термы, такие как *Forward* и *Turn(Right)*. В данном разделе предполагается, что среда содержит только одного агента. (Если количество агентов больше одного, может быть вставлен дополнительный параметр для указания на то, какой агент выполняет это действие.)
- **Ситуации** представляют собой логические термы, состоящие из начальной ситуации (обычно называемой  $S_0$ ) и всех ситуаций, которые создаются в результате применения некоторого действия в некоторой ситуации. Функция *Result(a, s)* (иногда называемая *Do*) обозначает ситуацию, возникающую в результате выполнения действия  $a$  в ситуации  $s$ . Эта идея иллюстрируется на рис. 10.2.

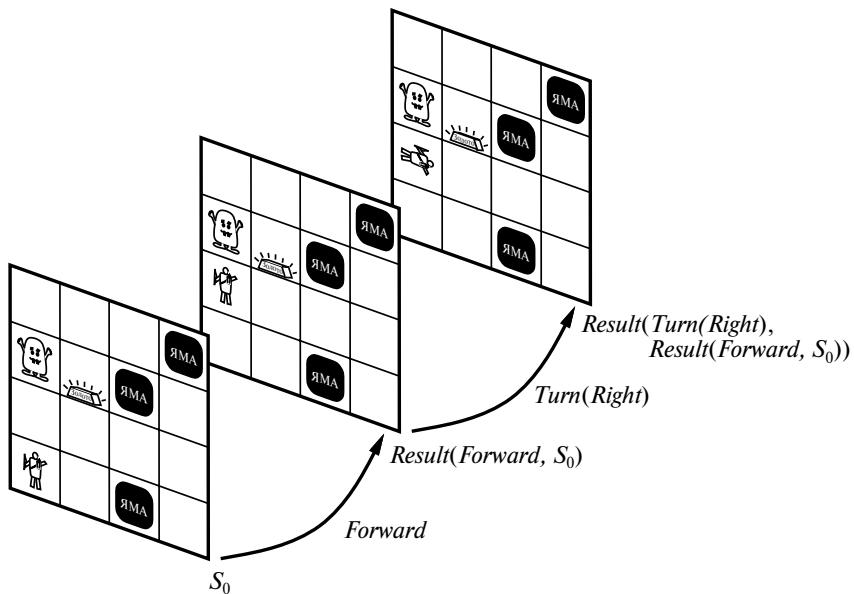


Рис. 10.2. Иллюстрация к понятию ситуации. В ситуационном исчислении каждая ситуация (кроме  $S_0$ ) представляет собой результат некоторого действия

- ❖ **Флюентными** называются функции и предикаты, которые изменяются от одной ситуации к другой, такие как местонахождение агента или наличие живого вампуса. Этот термин происходит от английского слова *fluent*, о котором в словаре сказано, что оно обозначает нечто текучее, как жидкость. В данном контексте термином “флюентный” обозначается все, что перетекает, или изменяется от одной ситуации к другой. В соответствии с общепринятым соглашением ситуация — это всегда последний параметр флюентного высказывания. Например, во флюентном высказывании  $\text{Holding}(G_1, S_0)$  сообщается о том, что агент не владеет золотом  $G_1$  в начальной ситуации  $S_0$ , а в высказывании  $\text{Age}(\text{Wumpus}, S_0)$  обозначается возраст вампуса в ситуации  $S_0$ .
- Допускается также использовать **вневременные**, или **неизменные** предикаты и функции. В качестве соответствующих примеров можно указать предикат  $\text{Gold}(G_1)$  и функцию  $\text{LeftLegOf}(\text{Wumpus})$ .

Кроме единственных действий, имеет также смысл рассуждать о последовательностях действий. Мы можем определять результаты последовательностей действий в терминах результатов отдельных действий. Прежде всего необходимо отметить, что выполнение пустой последовательности оставляет ситуацию неизменной:

$$\text{Result}([], s) = s$$

Выполнение непустой последовательности равносильно выполнению первого действия, а затем выполнению остальной ее части в результирующей ситуации:

$$\text{Result}([a|seq], s) = \text{Result}(seq, \text{Result}(a, s))$$

Агент, действующий на основе ситуационного исчисления, должен быть способен определять итог некоторой последовательности действий посредством логического вывода; в этом состоит задача **проектирования** (нахождения *прекции* действий на будущее). При наличии подходящего конструктивного алгоритма логического вывода он должен также обладать способностью находить последовательность действий, позволяющую достичь желаемого эффекта; в этом состоит задача **планирования**.

Мы будем использовать пример из модифицированной версии мира вампуса, в которой можно не задумываться об ориентации агента и в которой агент может переходить с помощью действия *Go* из одного места в соседнее. Предположим, что агент находится в квадрате  $[1, 1]$ , а золото — в квадрате  $[1, 2]$ . Задача состоит в том, чтобы перенести золото в квадрат  $[1, 1]$ . Флюентными предикатами являются *At(o, x, s)* и *Holding(o, s)*. В таком случае первоначальная база знаний может включать следующее описание:

$$\text{At}(\text{Agent}, [1, 1], S_0) \wedge \text{At}(G_1, [1, 2], S_0)$$

Но этого описания совершенно недостаточно, поскольку в нем не сказано о том, какие предикаты не являются истинными в состоянии  $S_0$ . (Дальнейшее обсуждение этой темы продолжается на с. 484.) Полное описание состоит в следующем:

$$\begin{aligned} \text{At}(o, x, S_0) &\Leftrightarrow [(o = \text{Agent} \wedge x = [1, 1]) \vee (o = G_1 \wedge x = [1, 2])] \\ &\neg \text{Holding}(o, S_0) \end{aligned}$$

Необходимо также указать, что  $G_1$  — золото и что квадраты  $[1, 1]$  и  $[1, 2]$  являются соседними:

$$\text{Gold}(G_1) \wedge \text{Adjacent}([1, 1], [1, 2]) \wedge \text{Adjacent}([1, 2], [1, 1])$$

Неплохо было бы иметь возможность доказать, что агент достигнет цели, перейдя в квадрат  $[1, 2]$ , схватив золото и возвратившись в квадрат  $[1, 1]$ , следующим образом:

$$\text{At}(G_1, [1, 1], \text{Result}([\text{Go}([1, 1], [1, 2]), \text{Grab}(G_1), \text{Go}([1, 2], [1, 1])], S_0))$$

Но гораздо интереснее составить план приобретения агентом золота и решать эту задачу, получая ответ на показанный ниже запрос, который словесно формулируется таким образом: “Какая последовательность действий *seq* приведет к тому, что золото окажется в квадрате  $[1, 1]$ ? ”

$$\exists \text{seq } \text{At}(G_1, [1, 1], \text{Result}(\text{seq}, S_0))$$

Рассмотрим, что должно войти в базу знаний, с помощью которой можно было бы получать ответы на такие запросы.

## Описание действий в ситуационном исчислении

В простейшей версии ситуационного исчисления каждое действие описывается с помощью двух аксиом: **аксиомы возможности**, которая указывает, существует или не существует возможность выполнить действие, и **аксиомы результата**, которая указывает, что произойдет после выполнения возможного действия. Мы будем использовать высказывание *Poss(a, s)* для обозначения того, что возможно выполнить действие *a* в ситуации *s*. Эти аксиомы имеют показанную ниже форму.

Аксиома возможности:

Предусловия  $\Rightarrow \text{Poss}(a, s)$

Аксиома результата:

$\text{Poss}(a, s) \Rightarrow$  Изменения, представляющие собой  
результат выполнения действия

Представим эти аксиомы для модифицированного мира вампуса. Чтобы можно было сократить соответствующие высказывания, мы будем исключать кванторы всеобщности, область действия которых распространяется на все высказывание. Предполагается, что переменная  $s$  пробегает по всем ситуациям,  $a$  — по действиям,  $o$  — по объектам (включая агентов),  $g$  — по состояниям владения и не владения золотом, а  $x$  и  $y$  — по местонахождениям.

Приведенные ниже аксиомы возможности для этого состояния мира указывают, что агент может переходить из одного местонахождения в другое, соседнее местонахождение, хватать кучу золота в текущем местонахождении и отпускать золото, которое он держит.

$$\begin{aligned} \text{At}(\text{Agent}, x, s) \wedge \text{Adjacent}(x, y) &\Rightarrow \text{Poss}(\text{Go}(x, y), s) \\ \text{Gold}(g) \wedge \text{At}(\text{Agent}, x, s) \wedge \text{At}(g, x, s) &\Rightarrow \text{Poss}(\text{Grab}(g), s) \\ \text{Holding}(g, s) &\Rightarrow \text{Poss}(\text{Release}(g), s) \end{aligned}$$

А в приведенных ниже аксиомах результата утверждается, что если действие возможно, то некоторые (флюентные) свойства будут иметь место в ситуации, возникающей в итоге выполнения данного действия. Переход из квадрата  $x$  в квадрат  $y$  приводит к пребыванию агента в квадрате  $y$ , схватывание золота приводит к владению золотом, а отпускание золота приводит к тому, что агент им больше не владеет.

$$\begin{aligned} \text{Poss}(\text{Go}(x, y), s) &\Rightarrow \text{At}(\text{Agent}, y, \text{Result}(\text{Go}(x, y), s)) \\ \text{Poss}(\text{Grab}(g), s) &\Rightarrow \text{Holding}(g, \text{Result}(\text{Grab}(g), s)) \\ \text{Poss}(\text{Release}(g), s) &\Rightarrow \neg \text{Holding}(g, \text{Result}(\text{Release}(g), s)) \end{aligned}$$

Сможем ли мы теперь доказать, что наш маленький план позволяет достичь цели, сформулировав все эти аксиомы? К сожалению, нет! Вначале все идет прекрасно; действие  $\text{Go}([1, 1], [1, 2])$  действительно возможно в ситуации  $S_0$  и аксиома результата для  $\text{Go}$  позволяет заключить, что агент достиг квадрата  $[1, 2]$ :

$\text{At}(\text{Agent}, [1, 2], \text{Result}(\text{Go}([1, 1], [1, 2]), S_0))$

Теперь рассмотрим действие  $\text{Grab}(G_1)$ . Необходимо показать, что оно возможно в новой ситуации, т.е. возможно следующее:

$\text{At}(G_1, [1, 2], \text{Result}(\text{Go}([1, 1], [1, 2]), S_0))$

К сожалению, в базе знаний нет ничего, что позволило бы обосновать такое заключение. Интуитивно можно понять, что действие агента  $\text{Go}$  не должно оказывать влияния на местонахождение золота, поэтому золото должно было все время находиться в квадрате  $[1, 2]$ , даже когда агент был в ситуации  $S_0$ .  *Проблема состоит в том, что аксиомы результата указывают, что изменилось, но не указывают, что осталось неизменным.*

Проблема представления всего того, что остается неизменным, называется  **проблемой окружения** (frame problem). Мы должны найти эффективное решение проблемы окружения, поскольку в реальном мире почти все и почти всегда остается

неизменным. В каждом действии затрагивается лишь крошечная часть всех флюентных высказываний.

Один из подходов состоит в том, что должны быть записаны явные **аксиомы окружения**, которые указывают, что остается неизменным. Например, после перемещения агента все другие объекты остаются на месте, если только он не берет эти объекты:

$$At(o, x, s) \wedge (o \neq Agent) \wedge \neg Holding(o, s) \Rightarrow At(o, x, Result(Go(y, z), s))$$

Если имеется  $F$  флюентных предикатов и  $A$  действий, то потребуется  $O(AF)$  аксиом окружения. С другой стороны, если бы каждое действие имело самое большое  $E$  результатов, где  $E$ , как правило, намного меньше, чем  $F$ , то необходимо было бы иметь возможность представлять, что происходит, с помощью гораздо меньшей базы знаний, с размером  $O(AE)$ . В этом и состоит **проблема представительного окружения** (representational frame problem). Тесно связанная с ней **проблема выводимого окружения** (inferential frame problem) заключается в том, что нужно проектировать результаты  $t$ -шаговой последовательности действий за время  $O(Et)$ , а не за время  $O(Ft)$  или  $O(AEt)$ , чтобы действовать успешно. Мы будем решать каждую из этих проблем по очереди. Но даже после этого останется еще одна проблема, связанная с тем, что должны быть обеспечены все необходимые условия для успешного выполнения любого действия. Например, действие *Go* окончится неудачей, если в ходе его выполнения агент погибнет. В этом состоит **проблема спецификации** (qualification problem), полное решение которой еще не найдено.

### Решение проблемы представительного окружения

Для решения проблемы представительного окружения достаточно лишь немного изменить точку зрения на то, как следует записывать аксиомы. Вместо регистрации результатов каждого действия мы будем рассматривать, как каждый флюентный предикат развивается во времени<sup>2</sup>. Применяемые при этом аксиомы называются **аксиомами состояния-преемника**. Они имеют следующую форму:

Аксиома состояния-преемника:

Действие возможно  $\Rightarrow$

(Флюентное высказывание  
является истинным в

результатирующем состоянии  $\Leftrightarrow$  Оно стало истинным в результате  
действия  $\vee$  оно было истинным и прежде, а действие оставило его  
неизменным)

Пояснив, что мы не рассматриваем невозможные действия, хотим обратить внимание читателя на то, что в приведенном выше определении используется логическая связка  $\Leftrightarrow$ , а не  $\Rightarrow$ . Таким образом, в этой аксиоме указано, что данное флюентное высказывание будет истинным тогда и только тогда, когда выполняется его правая часть. Иными словами, истинностное значение каждого флюентного высказывания в следующем состоянии определено как функция действия и истин-

---

<sup>2</sup> По сути именно этот подход был принят при создании агента на основе логической схемы в главе 7. И действительно, такие аксиомы, как уравнения 7.4 и 7.5, могут рассматриваться как аксиомы состояния-преемника.

ностного значения в текущем состоянии. Это означает, что следующее состояние полностью задано текущим состоянием и поэтому нет необходимости использовать дополнительные аксиомы окружения.

В аксиоме состояния-преемника для местонахождения агента утверждается, что агент находится в квадрате  $y$  после выполнения некоторого действия, либо если это действие было возможным и заключалось в перемещении в квадрат  $y$ , либо если агент уже находился в квадрате  $y$  и его действие не заключалось в том, чтобы перемещаться куда-то в другое место:

$$\begin{aligned} \text{Poss}(a, s) \Rightarrow \\ (\text{At}(\text{Agent}, y, \text{Result}(a, s)) \Leftrightarrow a = \text{Go}(x, y) \\ \vee (\text{At}(\text{Agent}, y, s) \wedge a \neq \text{Go}(y, z))) \end{aligned}$$

В аксиоме для флюентного предиката *Holding* утверждается, что агент владеет золотом  $g$  после выполнения некоторого действия, если этим действием было схватывание золота  $g$  и такое схватывание было возможно или если агент уже владел золотом  $g$  и выполненное агентом действие не заключалось в отпускании золота:

$$\begin{aligned} \text{Poss}(a, s) \Rightarrow \\ (\text{Holding}(g, \text{Result}(a, s)) \Leftrightarrow a = \text{Grab}(g) \\ \vee (\text{Holding}(g, s) \wedge a \neq \text{Release}(g))) \end{aligned}$$

 *Аксиомы состояния-преемника позволяют решить проблему представительного окружения*, поскольку общий размер этих аксиом измеряется величиной в  $O(AE)$  литералов: каждый из  $E$  результатов каждого из  $A$  действий упоминается один и только один раз. Литералы распределяются по  $F$  разным аксиомам, поэтому аксиомы имеют средний размер  $AE/F$ .

Внимательный читатель должен был заметить, что в этих аксиомах участвует флюентное высказывание *At*, касающееся агента, но не золота, поэтому мы все еще не можем доказать, что приведенный выше трехшаговый план позволяет достичь цели — переноса золота в квадрат [1, 1]. Мы должны указать, что  **неявным результатом** перемещения агента из квадрата  $x$  в квадрат  $y$  является то, что переместится также все золото, которое он несет (а также все муравьи, которые заползли в это золото, все пылинки и бактерии на этих муравьях и т.д.). Учет таких неявных результатов связан с решением так называемой  **проблемы распространения последствий** (ramification problem). Более подробно мы рассмотрим эту проблему позже, но для данной конкретной проблемной области ее можно решить, дописав более общую аксиому состояния-преемника для высказывания *At*. В новой аксиоме, которая обобщает предыдущую версию, утверждается, что объект  $o$  находится в квадрате  $y$ , либо если агент перешел в квадрат  $y$  и  $o$  — это или агент, или то, что держит агент, либо если объект  $o$  уже находился в квадрате  $y$  и агент никуда не переходил, притом что  $o$  является или агентом, или тем, что держит агент.

$$\begin{aligned} \text{Poss}(a, s) \Rightarrow \\ \text{At}(o, y, \text{Result}(a, s)) \Leftrightarrow (a = \text{Go}(x, y) \wedge (o = \text{Agent} \vee \text{Holding}(o, s))) \\ \vee (\text{At}(o, y, s) \wedge \neg(\exists z y \neq z \wedge a = \text{Go}(y, z) \\ \wedge (o = \text{Agent} \vee \text{Holding}(o, s)))) \end{aligned}$$

Возникает еще одна формальная сложность: процесс логического вывода, в котором используются эти аксиомы, должен позволять доказывать неравенства. Про-

стейшего рода неравенство задается между константами, например  $Agent \neq G_1$ . Общая семантика логики первого порядка допускает, чтобы разные константы ссылались на один и тот же объект, поэтому база знаний должна включать какую-то аксиому, позволяющую предотвратить такую ситуацию. В **аксиоме уникальных имен** (unique names axiom) провозглашается неравенство для каждой пары констант в базе знаний. Если же это условие подразумевается в программе автоматического доказательства теорем, а не записывается в базу знаний, то называется **предположением об уникальности имен** (unique names assumption). Необходимо также сформулировать условие неравенства между термами действий, например, утверждение о том, что  $Go([1, 1], [1, 2])$  — это действие, отличное от  $Go([1, 2], [1, 1])$  или  $Grab(G_1)$ . Прежде всего необходимо указать, что каждый тип действия является различным, т.е., например, что действие  $Go$  — это не действие  $Grab$ . Для каждой пары имен действий  $A$  и  $B$  необходимо иметь:

$$A(x_1, \dots, x_m) \neq B(y_1, \dots, y_n)$$

Затем требуется указать, что два терма действия с одним и тем же именем действия ссылаются на одно и то же действие тогда и только тогда, когда все участвующие в них объекты являются одинаковыми:

$$A(x_1, \dots, x_m) = A(y_1, \dots, y_n) \Leftrightarrow x_1 = y_1 \wedge \dots \wedge x_m = y_n$$

Все эти аксиомы, вместе взятые, называются **аксиомами уникального действия** (unique action axiom). Такая комбинация описания начального состояния, аксиом состояния-преемника, аксиом уникальности имен и аксиом уникального действия является достаточной для доказательства того, что предлагаемый план позволяет достичь цели.

### Решение проблемы выводимого окружения

Аксиомы состояния-преемника позволяют решить проблему представительного окружения, но не проблему выводимого окружения. Рассмотрим  $t$ -шаговый план  $p$ , такой, что  $S_t = Result(p, S_0)$ . Чтобы определить, какие флюентные высказывания являются истинными в состоянии  $S_t$ , необходимо рассмотреть каждую из  $F$  аксиом окружения в каждом из  $t$  временных шагов. Поскольку аксиомы имеют средний размер  $AE/F$ , это соответствует объему работы по логическому выводу, измеряемому величиной  $O(AEt)$ . Основной объем этой работы связан с копированием флюентных высказываний в неизменном виде из описания одной ситуации в описание следующей.

Для решения проблемы выводимого окружения можно воспользоваться следующими двумя подходами. Во-первых, можно отбросить ситуационное исчисление и изобрести новую формальную систему для записи аксиом. Указанная задача была решена в таких формальных системах, как **исчисление флюентных высказываний** (fluent calculus). Во-вторых, можно изменить механизм логического вывода таким образом, чтобы в нем аксиомы окружения обрабатывались более эффективно. На то, что должна существовать такая возможность, указывает сам факт, что трудоемкость этого простого подхода оценивается величиной  $O(AEt)$ ; но почему она должна зависеть от количества действий  $A$ , если точно известно, какое именно действие выполняется в каждом временном шаге? Чтобы определить, как можно улучшить состояние дел, вначале рассмотрим формат аксиом окружения:

$$\begin{aligned} \text{Poss}(a, s) \Rightarrow \\ F_i(\text{Result}(a, s)) \Leftrightarrow (a = A_1 \vee a = A_2 \dots) \\ \quad \vee F_i(s) \wedge (a \neq A_3) \wedge (a \neq A_4) \dots \end{aligned}$$

Таким образом, в каждой аксиоме рассматриваются несколько действий, которые могут сделать это флюентное высказывание истинным, и несколько действий, которые могут сделать его ложным. Мы можем формализовать такие истинностные преобразования, введя предикат  $\text{PosEffect}(a, F_i)$ , означающий, что в результате действия  $a$  флюентное высказывание  $F_i$  становится истинным, и предикат  $\text{NegEffect}(a, F_i)$ , означающий, что после выполнения действия  $a$  флюентное высказывание  $F_i$  становится ложным. Это означает, что приведенную выше систему аксиом можно переписать следующим образом:

$$\begin{aligned} \text{Poss}(a, s) \Rightarrow \\ F_i(\text{Result}(a, s)) \Leftrightarrow \text{PosEffect}(a, F_i) \vee [F_i(s) \wedge \neg \text{NegEffect}(a, F_i)] \\ \text{PosEffect}(A_1, F_i) \\ \text{PosEffect}(A_2, F_i) \\ \text{NegEffect}(A_3, F_i) \\ \text{NegEffect}(A_4, F_i) \end{aligned}$$

Ответ на вопрос о том, может ли такая операция формирования системы аксиом быть выполнена автоматически, зависит от точного формата аксиом окружения. Для того чтобы обеспечить использование подобных аксиом в эффективной процедуре логического вывода, необходимо провести три описанных ниже преобразования.

1. Проиндексировать предикаты  $\text{PosEffect}$  и  $\text{NegEffect}$  по их первому параметру таким образом, чтобы после получения информации о действии, которое произошло в момент времени  $t$ , можно было бы найти его результаты за время  $O(1)$ .
2. Проиндексировать аксиомы таким образом, чтобы после определения того, что  $F_i$  представляет собой результат какого-то действия, мы могли найти аксиому для  $F_i$  за время  $O(1)$ . В таком случае не придется даже рассматривать аксиомы, касающиеся тех флюентных высказываний, которые не являются результатом данного действия.
3. Представить каждую ситуацию как предыдущую ситуацию плюс некоторая дельта. Таким образом, если ничего не изменяется от одного шага к другому, то не нужно вообще выполнять никакой работы. При использовании старого подхода требовалось выполнять объем работы  $O(F)$ , вырабатывая утверждение для каждого флюентного высказывания  $F_i(\text{Result}(a, s))$  из предыдущих  $F_i(s)$  утверждений.

Поэтому в каждом временном шаге мы рассматриваем текущее действие, собираем данные о его результатах и обновляем множество истинных флюентных высказываний. Среднее количество таких обновлений в каждом временном шаге соответствует  $E$ , поэтому общая сложность измеряется величиной  $O(Et)$ . В этом и состоит решение проблемы выводимого окружения.

## Исчисление времени и событий

Ситуационное исчисление вполне себя оправдывает, если существует единственный агент, выполняющий мгновенные, дискретные действия, а если действия имеют продолжительность и могут накладываться друг на друга, то ситуационное исчисление становится довольно громоздким. Поэтому мы будем рассматривать данные темы с помощью альтернативной формальной системы, известной под названием **исчисление событий** (event calculus), которая основана на точках во времени, а не на ситуациях. (Термины “событие” и “действие” могут использоваться как взаимозаменяемые. Неформально выражаясь, “событие” обозначает более широкий класс действий, включая те, в которых нет явного агента. С описанием такого класса проще справиться с помощью исчисления событий, а не ситуационного исчисления.)

В исчислении событий флюентные высказывания становятся истинными или ложными в определенных точках во времени, а не в ситуациях, притом что само исчисление предназначено для того, чтобы с его помощью можно было формировать рассуждения, касающиеся интервалов времени. В аксиоме исчисления событий указано, что флюентное высказывание является истинным в какой-то момент времени, если действие, описанное этим высказыванием, было инициировано некоторым событием в некоторый момент времени в прошлом и еще не закончено под влиянием какого-либо промежуточного события. Отношения *Initiates* и *Terminates* играют примерно такую же роль, как и отношение *Result* в ситуационном исчислении; отношение *Initiates*( $e, f, t$ ) означает, что возникновение события  $e$  во время  $t$  вызвало то, что флюентное высказывание  $f$  стало истинным, а отношение *Terminates*( $w, f, t$ ) означает, что высказывание  $f$  перестало быть истинным. Мы будем использовать предикат *Happens*( $e, t$ ) для обозначения того, что событие  $e$  произошло во время  $t$ , и предикат *Clipped*( $f, t, t_2$ ) для обозначения того, что высказывание  $f$  перестало быть истинным под влиянием некоторого события, прошедшего в какое-то время между  $t$  и  $t_2$ . Формально эта аксиома записывается следующим образом:

Аксиома исчисления событий:

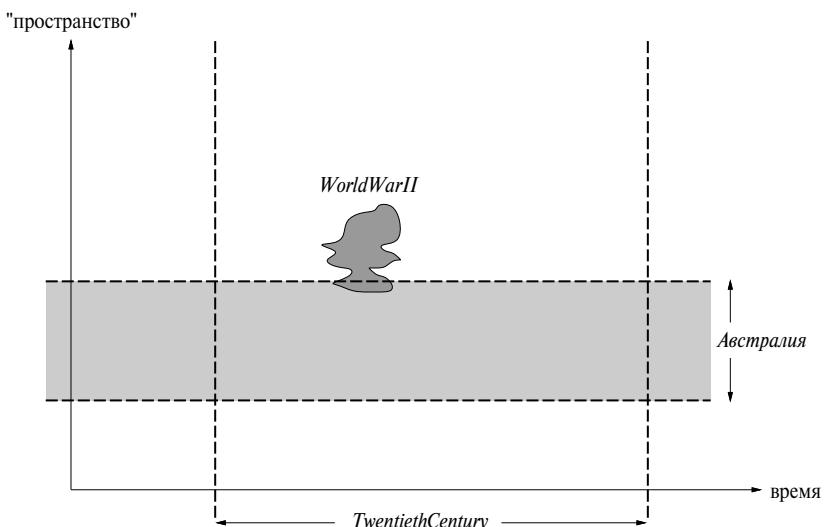
$$\begin{aligned} T(f, t_2) \Leftrightarrow & \exists e, t \ Happens(e, t) \wedge \text{Initiates}(e, f, t) \wedge (t < t_2) \\ & \wedge \neg \text{Clipped}(f, t, t_2) \\ Clipped(f, t, t_2) \Leftrightarrow & \exists e, t_1 \ Happens(e, t_1) \wedge \text{Terminates}(e, f, t_1) \\ & \wedge (t < t_1) \wedge (t_1 < t_2) \end{aligned}$$

Это позволяет нам получить такие же функциональные возможности, как и с использованием ситуационного исчисления, в сочетании с возможностями формировать утверждения о точках и интервалах времени, что позволяет, например, сформулировать высказывание *Happens*(*TurnOff(LightSwitch<sub>1</sub>)*, 1:00), чтобы сообщить о выключении света точно в 1:00.

Было разработано много дополнений к исчислению событий, чтобы можно было решать задачи с опосредованными событиями; событиями, имеющими продолжительность; событиями, происходящими одновременно; непрерывно изменяющимися событиями; недетерминированными результатами; причинно-следственными ограничениями и другими осложнениями. Мы вернемся к рассмотрению этих тем в следующем подразделе. Следует откровенно сказать, что в данный момент для большинства таких задач еще отсутствуют полностью удовлетворительные решения, но на пути к достижению этих решений не обнаружено каких-либо непреодолимых препятствий.

## Обобщенные события

До сих пор в этой главе рассматривались две основные концепции: действия и объекты, а теперь настало время перейти к описанию того, как эти концепции укладываются во всеобъемлющую онтологию, в которой и действия, и объекты могут трактоваться как аспекты физического универсума. Мы рассматриваем конкретный универсум как имеющий и пространственное, и временное измерения. Мир вампуса характеризуется пространственным компонентом, представленным в виде двухмерной решетки, а время в нем дискретно; мир, в котором существуют люди, имеет три пространственных измерения и одно временное<sup>3</sup>, причем все эти измерения непрерывны. **Обобщенное событие** (generalized event) состоит из аспектов некоторого “пространственно-временного фрагмента” — части этого многомерного пространственно-временного универсума. Эта абстракция позволяет обобщить большинство концепций, рассматривавшихся нами до сих пор, включая действия, местонахождения, интервалы времени, флюентные высказывания и физические объекты. Иллюстрация этой основной идеи приведена на рис. 10.3. Начиная с этого момента, мы будем использовать неуточненный термин “событие” для обозначения обобщенных событий.



*Рис. 10.3. Обобщенные события. Универсум имеет три пространственных и одно временное измерения; на этом рисунке показано только одно пространственное измерение. Все события являются частью, PartOf, этого универсума. Любое событие, такое как Вторая мировая война, WorldWarII, происходит в части пространства-времени с немного произвольными и изменяющимися во времени границами. Любой интервал Interval, такой как двадцатый век, TwentiethCentury, имеет фиксированную, ограниченную временную протяженность и максимальную пространственную протяженность, а любое место, Place, такое как Австралия, имеет примерно фиксированную пространственную протяженность и максимальную временную протяженность*

<sup>3</sup> Некоторые физики, работающие в области теории суперструн (string theory), доказывают, что измерений — 10 или больше, а другие утверждают, что мир дискретен, но для формирования обычных рассуждений вполне подходит представление в виде четырехмерного непрерывного пространства/времени.

Например, Вторая мировая война — это событие, которое имело место в различных точках пространства-времени, что обозначено на этом рисунке в виде серого пятна неправильной формы. Это событие можно разбить на ряд подсобытий<sup>4</sup>:

```
SubEvent(BattleOfBritain, WorldWarII)
```

Равным образом, Вторая мировая война — это подсобытие двадцатого столетия:

```
SubEvent(WorldWarII, TwentiethCentury)
```

Двадцатое столетие — это интервал времени. Интервалы представляют собой фрагменты пространства-времени, которые включают все пространство между двумя точками во времени. Функция *Period(e)* обозначает наименьший интервал, включающий событие *e*, а функция *Duration(i)* — это продолжительность времени, занятая некоторым интервалом *i*, поэтому можно сформулировать высказывание *Duration(Period(WorldWarII)) > Years(5)*.

Австралия — это место, т.е. фрагмент пространства-времени с некоторымификсированными пространственными границами. Границы могут изменяться во времени вследствие геологических или политических изменений. Для обозначения отношения с участием подсобытия, которое имеет место, если пространственная проекция одного события является частью *PartOf* другого события, используется предикат *In*:

```
In(Sydney, Australia)
```

Функция *Location(e)* обозначает наименьшее место, которое включает событие *e*.

Как и объекты любого другого рода, события могут быть сгруппированы по категориям. Например, *WorldWarII* принадлежит к категории войн *Wars*. Чтобы сформулировать утверждение, что гражданская война происходила в Англии в 1640-х годах, можно привести следующее высказывание:

```
 $\exists w \ w \in CivilWars \wedge SubEvent(w, 1640s) \wedge In(Location(w), England)$ 
```

Понятие категории событий позволяет найти ответ на вопрос, которого мы избегали при описании результатов действий в разделе 10.3: на что именно ссылаются такие логические термы, как *Go([1, 1], [1, 2])*? Являются ли они событиями? Ответ на этот вопрос отрицателен, что на первый взгляд может показаться неожиданным. Но в этом можно убедиться, рассмотрев план с двумя “идентичными” действиями, такой как следующий:

```
[Go([1, 1], [1, 2]), Go([1, 2], [1, 1]), Go([1, 1], [1, 2])]]
```

В этом плане выражение *Go([1, 1], [1, 2])* не может быть именем события, поскольку в нем представлены два разных события, происходящих в разное время. Вместо этого *Go([1, 1], [1, 2])* представляет собой имя категории событий — всех тех событий, в которых агент переходит из квадрата  $[1, 1]$  в квадрат  $[1, 2]$ . Приведенный выше трехшаговый план сообщает о том, что должны осуществиться экземпляры этих трех категорий событий.

Обратите внимание на то, что здесь мы впервые встретились с категориями, именованными с помощью сложных термов, а не просто константных символов. Это обстоятельство не должно стать источником новых затруднений, поскольку фактически мы

---

<sup>4</sup> Обратите внимание на то, что отношение *SubEvent* (Подсобытие) — это частный случай отношения *PartOf* и также является транзитивным и рефлексивным.

можем использовать такую структуру параметров предикатов, которая является наиболее удобной. Устранение параметров позволяет создать более общую категорию:

$$Go(x, y) \subseteq GoTo(y)$$

$$Go(x, y) \subseteq GoFrom(x)$$

Аналогичным образом можно добавлять параметры для создания более конкретных категорий. Например, чтобы описать действия, осуществляемые другими агентами, можно добавить параметр с обозначением агента. Поэтому, чтобы сформулировать утверждение, что знаменитый математик Шанкар вчера прилетел из Нью-Йорка в Нью-Дели, можно записать следующее:

$$\exists e \ e \in Fly(Shankar, NewYork, NewDelhi) \wedge SubEvent(e, Yesterday)$$

Формулы в таком виде применяются настолько часто, что мы создадим для таких формул сокращение  $E(c, i)$ , которое будет означать, что элемент категории событий  $c$  является подсобытием события или интервала  $i$ :

$$E(c, i) \Leftrightarrow \exists e \ e \in c \wedge SubEvent(e, i)$$

Таким образом, переформулируем приведенное выше утверждение следующим образом:

$$E(Fly(Shankar, NewYork, NewDelhi), Yesterday)$$

## Процессы

События, рассматривавшиеся нами до сих пор, были тем, что принято называть **дискретными событиями** — они имеют определенную структуру. Путешествие Шанкара имело начало, середину и конец. Если бы он прервал свое путешествие на полпути, то связанное с этим событие было бы другим — не путешествием из Нью-Йорка в Нью-Дели, а, допустим, поездкой из Нью-Йорка в какой-то европейский город. С другой стороны, категория событий, обозначенная высказыванием  $Flying(Shankar)$ , имеет другое качество. Если бы мы стали исследовать небольшой интервал полета Шанкара, скажем, третий двадцатиминутный отрезок времени (в течение которого он нетерпеливо ждал, когда ему принесут еще один пакетик арахиса), это событие все еще оставалось бы элементом события  $Flying(Shankar)$ . В действительности такое утверждение остается истинным для любого субинтервала.

Категории событий, обладающих этим свойством, называются категориями **процессов** (process) или категориями **вневременных событий** (liquid event). Любой субинтервал процесса также является элементом той же категории процессов. Чтобы высказать мысль, что Шанкар, например, был вчера в полете в какой-то момент времени, можно использовать такую же систему обозначений, которая используется для дискретных событий:

$$E(Flying(Shankar), Yesterday)$$

Кроме того, часто возникает необходимость сформулировать утверждение, что некоторый процесс происходил на протяжении всего определенного временного интервала, а не просто некоторого его субинтервала. Для этого используется предикат  $T$  следующим образом:

$$T(Working(Stuart), TodayLunchHour)$$

Высказывание  $T(c, i)$  обозначает, что некоторое событие типа  $c$  происходило точно в течение интервала  $i$ , т.е. что это событие началось и закончилось в то же время, что и сам этот интервал.

Различие между вневременными (liquid) и внутривременными (nonliquid) событиями полностью аналогично различию между веществами (или материей) и отдельными объектами. В действительности некоторые ученые называют типы вневременных событий  **временными веществами** (temporal substance), по аналогии с тем, что масло рассматривается как **пространственное вещество** (spatial substance).

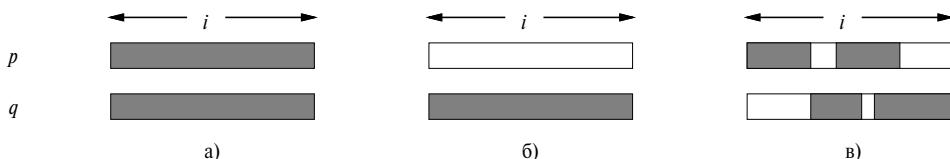
Концепция вневременных событий позволяет описывать не только процессы непрерывного изменения, но и процессы непрерывного “неизменения”. Такие процессы часто называют  **состояниями** (state). Например, “Шанкар находится в Нью-Йорке” — это категория состояний, для которой нами принято обозначение  $In(Shankar, NewYork)$ . Чтобы сформулировать утверждение, что Шанкар находился весь нынешний день в Нью-Йорке, можно записать следующее:

$$T(In(Shankar, NewYork), Today)$$

Для формирования более сложных высказываний о состояниях и событиях могут применяться сочетания примитивных высказываний. Этот подход называется  **исчислением флюентных высказываний** (fluent calculus). В исчислении флюентных высказываний используются комбинации флюентных высказываний, а не просто отдельные флюентные высказывания. Выше уже был показан способ представления события, состоящего из двух событий, происходящих одновременно, а именно функция  $Both(e_1, e_2)$ . В исчислении флюентных высказываний такое выражение обычно записывается сокращенно с помощью инфиксной системы обозначений:  $e_1 \circ e_2$ . Например, чтобы сформулировать утверждение, что некто шел и одновременно жевал резинку, можно записать следующее:

$$\exists p, i \ (p \in People) \wedge T(Walk(p) \circ ChewGum(p), i)$$

Функция  $\circ$  является коммутативной и ассоциативной, как и логическая конъюнкция. Могут быть также определены аналоги дизъюнкции и отрицания, но необходимо соблюдать осторожность, поскольку существуют два одинаково приемлемых способа интерпретации дизъюнкции событий. Например, утверждение, что “в течение последних двух минут агент либо шел, либо жевал резинку”, может означать, что агент либо выполнял одно или другое из этих действий в течение всего интервала времени, либо чередовал эти два действия. Для обозначения таких двух вариантов будут использоваться предикаты *OneOf* и *Either*. Сложные события схематически показаны на рис. 10.4.



*Рис. 10.4. Схематическая иллюстрация сложных событий: одновременное протекание двух событий  $T(Both(p, q), i)$ , обозначаемое также как  $T(p \circ q, i)$  (а); осуществление одного из двух событий  $T(OneOf(p, q), i)$  (б); чередование двух событий  $T(Either(p, q), i)$  (в)*

## Интервалы

Время — это важный фактор для любого агента, выполняющего действия, и поэтому в области представления временных интервалов был проведен большой объем исследовательских работ. В данном разделе рассматриваются интервалы двух типов: моменты времени и продолжительные интервалы. Различие между ними состоит в том, что только моменты времени имеют нулевую продолжительность:

```
Partition({Moments, ExtendedIntervals}, Intervals)
i ∈ Moments ⇔ Duration(i) = Seconds(0)
```

Затем необходимо ввести временную шкалу и связать точки на этой шкале с моментами времени, что позволяет сформулировать понятие абсолютных значений времени. Временная шкала выбирается произвольно; в данной книге время измеряется в секундах и используется соглашение, что момент времени в полночь (среднее время по Гринвичу) 1 января 1900 года имел значение времени 0. Функции *Start* и *End* позволяют определить самый первый и самый последний моменты времени в интервале, а функция *Time* сообщает момент времени на временной шкале, соответствующий текущему моменту. Функция *Duration* измеряет разность между временем окончания и временем начала. Примеры применения этих функций приведены ниже.

```
Interval(i) ⇒ Duration(i) = (Time(End(i)) - Time(Start(i)))
Time(Start(AD1900)) = Seconds(0)
Time(Start(AD2001)) = Seconds(3187324800)
Time(End(AD2001)) = Seconds(3218860800)
Duration(AD2001) = Seconds(31536000)
```

Для того чтобы было проще читать эти числа, обозначающие количество секунд от начала отсчета, введем также функцию *Date*, которая принимает шесть параметров (часы, минуты, секунды, день, месяц и год) и возвращает точку во времени:

```
Date(Start(AD2001)) = Date(0, 0, 0, 1, Jan, 2001)
Date(0, 20, 21, 24, 1, 1995) = Seconds(3000000000)
```

Предикат *Meet* позволяет определить, равно ли время окончания первого интервала времени начала второго интервала; эти значения времени задаются в секундах. Существует возможность определить такие предикаты, как *Before*, *After*, *During* и *Overlap*, исключительно в терминах предиката *Meet*, но более интуитивно понятными являются их определения в терминах точек на временной шкале (графическое представление этих предикатов приведено на рис. 10.5):

```
Meet(i, j) ⇔ Time(End(i)) = Time(Start(j))
Before(i, j) ⇔ Time(End(i)) < Time(Start(j))
After(j, i) ⇔ Before(i, j)
During(i, j) ⇔ Time(Start(j)) ≤ Time(Start(i))
          ∧ Time(End(i)) ≤ Time(End(j))
Overlap(i, j) ⇔ ∃k During(k, i) ∧ During(k, j)
```

Например, чтобы сформулировать утверждение, что царствование Елизаветы II следовало за царствованием Георга VI, а царствование Элвиса Пресли совпадало с периодом 1950-х годов, можно записать следующее:

```
After(ReignOf(ElizabethII), ReignOf(GeorgeVI))
Overlap(Fifties, ReignOf(Elvis))
```

$Start(Fifties) = Start(AD1950)$   
 $End(Fifties) = End(AD1959)$

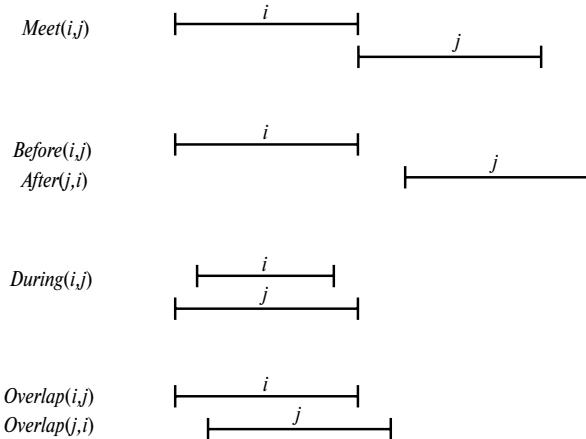


Рис. 10.5. Предикаты, задаваемые на временных интервалах

### Флюентные высказывания и объекты

Как было указано выше, физические объекты могут рассматриваться как обобщенные события в том смысле, что любой физический объект представляет собой фрагмент пространства–времени. Например, США можно рассматривать как событие, которое началось, скажем, в 1776 году в виде союза 13 штатов и все еще продолжается в наши дни как союз 50 штатов. Мы можем описать изменяющиеся свойства объекта *USA* с использованием флюентных высказываний, касающихся состояния. Например, можно сформулировать утверждение, что в какой-то момент в 1999 году население США составляло 271 миллион человек:

$E(Population(USA, 271000000), AD1999)$

Еще одним свойством объекта *USA*, которое изменяется через каждые четыре или восемь лет, если не учитывать непредвиденные происшествия, является смена президента этого государства. Можно было бы предложить описывать это свойство, условившись считать, что *President(USA)* — это логический терм, который обозначает различные объекты в разное время. К сожалению, это невозможно, поскольку в каждой конкретной структуре модели любой терм обозначает один и только один объект. (Терм *President(USA, t)* мог бы обозначать разные объекты в зависимости от значения  $t$ , но в принятой нами онтологии временные индексы рассматриваются отдельно от флюентных высказываний.) Единственная возможность состоит в том, чтобы считать, что *President(USA)* обозначает единственный объект, который в разное время состоит из разных людей. Это — объект, которым был Джордж Вашингтон с 1789 по 1796 гг., Джон Адамс с 1796 по 1800 гг. и т.д. (рис. 10.6).

Для того чтобы высказать утверждение, что Джордж Вашингтон был президентом в течение всего 1790 года, можно записать следующее:

$T(President(USA)) = GeorgeWashington, AD1790$

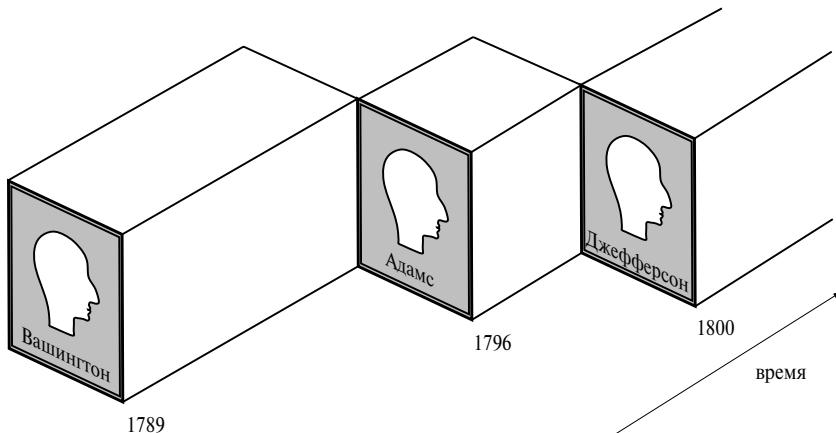


Рис. 10.6. Схематическое изображение объекта *President (USA)* за первые 15 лет его существования

Однако необходимо соблюдать осторожность. В этом высказывании знак = должен представлять собой функциональный символ, а не стандартный логический оператор. Его интерпретация состоит не в том, что *GeorgeWashington* и *President (USA)* были логически идентичными в 1790 году; логическое равенство есть нечто, неизменное во времени. В данном случае логическое равенство существует между подсобытиями каждого объекта, которые определены на период 1790 года.

Кроме того, не следует путать физический объект *GeorgeWashington* с коллекцией атомов, из которых он состоял. Джордж Вашингтон не был логически равен какой-либо конкретной коллекции атомов, поскольку множество атомов, из которых он состоял, существенно изменялось во времени. Жизнь его была коротка, а каждый атом, отдельно взятый, имеет свою собственную, очень длинную продолжительность жизни. На какой-то период времени они пересеклись, и тогда этот временной срез периода существования данного атома стал частью (*PartOf*) Джорджа Вашингтона, после чего каждый из этих двух объектов направился по своему пути.

## 10.4. МЫСЛИТЕЛЬНЫЕ СОБЫТИЯ И МЫСЛИМЫЕ ОБЪЕКТЫ

Агенты, которые были созданы в этой книге до сих пор, были уверены в истинности или ложности некоторых высказываний и обладали способностью выводить логическим путем новые убеждения, но ни один из этих агентов не обладал какими-либо знаниями о самих убеждениях или о логическом выводе. Но в одноагентных проблемных областях знания о собственных знаниях и процессах формирования рассуждений весьма полезны для управления логическим выводом. Например, если некто знает, что он ничего не знает о географии Румынии, то для него нет необходимости впустую затрачивать невероятные вычислительные усилия, чтобы рассчитать кратчайший путь от Арада до Бухареста. Вместо этого он может порассуждать о собственных знаниях, чтобы составить планы их расширения, например, купив карту Румынии. Кроме того, в мультиагентных проблемных областях для агента важно иметь возможность рассуждать о мыслимых состояниях других агентов. Например,

наилучший способ попасть в Бухарест вполне может знать румынский полицейский, поэтому агенту достаточно обратиться к нему за помощью.

По сути нам требуется модель мыслимых объектов, которые находятся в чьей-то голове (или базе знаний), и мыслительных процессов, которые манипулируют с этими мыслимыми объектами. Такая модель должна быть достоверной, но не требуется, чтобы она была слишком детализированной. Нам не нужно иметь способность предсказывать, сколько именно миллисекунд потребуется, чтобы какой-то конкретный агент пришел к нужному логическому заключению, или же прогнозировать, какие именно нейроны активизируются, когда исследуемое животное столкнется с конкретными визуальными стимулами. Для нас будет достаточно иметь возможность прийти к выводу, что румынский полицейский сообщит нам, как попасть в Бухарест, если будет знать дорогу и убедится в том, что мы действительно заблудились, а не подшучиваем над ним.

## Формальная теория убеждений

Начнем с изучения связей между агентами и “мыслимыми объектами”, таких как *Believes* (Убежден), *Knows* (Знает) и *Wants* (Желает). Отношения такого рода называются **пропозициональными позициями** (propositional attitude), поскольку они описывают позицию, которую может занять агент по отношению к некоторому высказыванию. Предположим, что Лойс в чем-то убеждена, т.е. *Believes(Lois, x)*. Какого рода объектом является *x*? Очевидно, что *x* не может быть логическим высказыванием. Если утверждение, что Супермен летает (*Flies(Superman)*) — логическое высказывание, то нельзя составить выражение *Believes(Lois, Flies(Superman))*, поскольку параметрами предикатов могут быть только термы (а не высказывания). Но если *Flies* — функция, то *Flies(Superman)* становится подходящим кандидатом на использование в качестве мыслимого объекта, а *Believes* может быть отношением между агентом и пропозициональным флюентным термом. Преобразование высказывания в объект называется **овеществлением** (reification)<sup>5</sup>.

По-видимому, такой подход позволяет достичь желаемого — появления у агента способности рассуждать об убеждениях агентов. Но, к сожалению, при использовании этого подхода возникает определенная проблема: если киноактер Кент Кларк и его герой Супермен — это одно и то же (т.е. *Clark=Superman*), то полет Кларка и полет Супермена — одна и та же категория событий, т.е. *Flies(Clark)=Flies(Superman)*. Поэтому мы обязаны сделать такой вывод: если Лойс уверена в том, что Супермен может летать, она должны быть также уверена в том, что Кларк может летать, даже если она не уверена в том, что Кларк — Супермен, как показано ниже.

$$\begin{aligned} (\text{Superman} = \text{Clark}) &= \\ (\text{Believes}(\text{Lois}, \text{Flies}(\text{Superman}))) &\Leftrightarrow \text{Believes}(\text{Lois}, \text{Flies}(\text{Clark})) \end{aligned}$$

В определенном смысле такое утверждение недалеко от истины: Лойс убеждена в том, что есть некое лицо, которому некогда присвоили имя Кларк, и что это лицо может летать. Но есть еще один смысл, в котором приведенное выше высказывание является ложным: если Лойс задать вопрос: “Может ли Кларк летать?”, она, безус-

<sup>5</sup> Англоязычный вариант термина “овеществление”, reification, происходит от латинского слова *res*. Джон Маккарти предложил термин-кальку “thingification”, но последний так и не прижился.

ловно, ответит отрицательно. Овеществленные объекты и события вполне приемлемы для использования в первом смысле толкования предиката *Believes*, но во втором толковании необходимо овеществлять описания этих объектов и событий таким образом, чтобы *Clark* и *Superman* могли иметь разные описания (даже несмотря на то, что оба эти имени относятся к одному и тому же объекту).

Формально свойство высказывания, позволяющее свободно подставлять любой терм вместо равного терма, называется **ссылочной прозрачностью** (referential transparency). В логике первого порядка свойством ссылочной прозрачности обладает каждое отношение. Но нам было бы желательно иметь возможность определять *Believes* (и другие пропозициональные позиции) как отношения, второй параметр которых является ссылочно **непрозрачным** (opaque), т.е. таким, чтобы без изменения смысла нельзя было бы подставить равный терм вместо второго параметра.

Существуют два способа достижения этой цели. Первый из них состоит в использовании другой формы логики, называемой **модальной логикой** (modal logic), в которой такие пропозициональные позиции, как *Believes* и *Knows*, становятся **модальными операторами**, обладающими свойством ссылочной непрозрачности. Этот подход рассматривается в разделе с историческими заметками. Второй подход, который будет применяться в данной главе, состоит в эффективном обеспечении непрозрачности в рамках ссылочно прозрачного языка с использованием **синтаксической теории** мыслимых объектов. Это означает, что мыслимые объекты представляются константами — **строками**. Результатом становится грубая модель базы знаний агента, которая представлена как состоящая из строк, соответствующих высказываниям, в истинности которых убежден агент. Стока — это сложный терм, обозначающий список символов, поэтому событие *Flies(Clark)* может быть представлено в виде списка символов  $[F, l, i, e, s, (, C, l, a, r, k, )]$ , которые мы будем сокращенно записывать в виде строки, заключенной в кавычки, "Flies(Clark)". Эта синтаксическая теория включает **аксиому об уникальности строк**, в которой утверждается, что строки являются идентичными тогда и только тогда, когда они состоят из идентичных символов в одинаковой последовательности. Таким образом, даже если *Clark=Superman*, мы все равно имеем "*Clark*" ≠ "*Superman*".

Теперь достаточно лишь предусмотреть синтаксис, семантику и теорию доказательства для языка представления строк, точно так же, как это было сделано в главе 7. Различие состоит в том, что необходимо определить все эти компоненты в логике первого порядка. Начнем с определения *Den* (сокращение от *denotatum* — денотат, или обозначаемое) как функции, которая отображает некоторую строку на обозначаемый ею объект, и *Name* как функции, которая отображает объект на строку, представляющую собой имя константы, обозначающей этот объект. Например, денотатом строки "*Clark*" и строки "*Superman*" является объект, на который ссылается константный символ *ManOfSteel* (Человек из стали), а именем этого объекта в базе знаний может быть "*Superman*", "*Clark*" или какая-то другая константа, например "*X<sub>11</sub>*".

```
Den("Clark") = ManOfSteel ∧ Den("Superman") = ManOfSteel
Name(ManOfSteel) = "X11"
```

Следующий этап состоит в том, чтобы определить правила логического вывода для логических агентов. Например, может потребоваться сформулировать утверждение, что любой логический агент способен использовать правило отделения: если он

уверен в истинности высказываний  $p$  и  $p \Rightarrow q$ , то он будет также уверен в истинности  $q$ . Первая попытка записать эту аксиому может состоять в следующем:

$$\text{LogicalAgent}(a) \wedge \text{Believes}(a, p) \wedge \text{Believes}(a, "p \Rightarrow q") \Rightarrow \\ \text{Believes}(a, q)$$

Но эта попытка неудачна, поскольку строка " $p \Rightarrow q$ " содержит буквы ' $p$ ' и ' $q$ ', но не имеет ничего общего со строками, являющимися значениями переменных  $p$  и  $q$ . Правильная формулировка состоит в следующем:

$$\text{LogicalAgent}(a) \wedge \text{Believes}(a, p) \wedge \text{Believes}(a, \text{Concat}(p, "\Rightarrow", q)) \\ \Rightarrow \text{Believes}(a, q)$$

где  $\text{Concat}$  — функция на строках, которая соединяет (конкатенирует) свои параметры. Мы будем сокращенно записывать операцию  $\text{Concat}(p, "\Rightarrow", q)$  с помощью подчеркивания, как " $\underline{p} \Rightarrow \underline{q}$ ". Это означает, что вхождение  $\underline{x}$  в строке должно быть **взятым без кавычек**, или **раскавыченным**, т.е. мы должны подставить строковое значение переменной  $x$ . В языке Lisp такая операция выполняется с помощью оператора “запятая/обратная кавычка”, а в языке Perl для этого используется интерполяция \$-переменной.

После введения других правил логического вывода, кроме правила отделения, мы получим возможность отвечать на вопросы в такой форме: “Если дано, что логический агент знает такие-то предпосылки, может ли он прийти к такому-то заключению?” Кроме этих обычных правил логического вывода, требуются определенные правила, характерные для рассуждений об убеждениях. Например, следующее правило утверждает, что если логический агент в чем-то уверен, то он уверен, что он в этом уверен:

$$\text{LogicalAgent}(a) \wedge \text{Believes}(a, p) \Rightarrow \text{Believes}(a, \text{Believes}(\text{Name}(a), \underline{p}))$$

Теперь, согласно нашим аксиомам, любой агент может безошибочно вывести любое следствие из своих убеждений. Такое свойство агента называется **логическим всеведением** (logical omniscience). Было сделано много попыток определить спецификации ограниченных рациональных агентов, которые способны выполнять ограниченное количество логических выводов за ограниченное время. Ни одна из этих попыток не оказалась полностью успешной, но описанные выше формулировки позволяют получить в высшей степени обоснованный ряд предсказаний о возможностях ограниченных агентов.

## Знания и убеждения

Соотношение между знаниями и убеждениями широко исследовалось в философии. Общепризнано, что знания — это убеждения, истинность которых подтвердилась. Таким образом, если вы в чем-то уверены по неоспоримо надежным причинам и если то, в чем вы уверены, действительно истинно, то вы это знаете. Условие, согласно которому причина должна быть “неоспоримо надежной”, необходимо для предотвращения такой ситуации, что вы будете утверждать: “Я знаю, что эта монета всегда будет падать орлом вверх”, и окажетесь правы лишь применительно к половине бросков монеты.

Допустим, что предикат  $\text{Knows}(a, p)$  означает следующее: агент  $a$  знает, что высказывание  $p$  является истинным. Возможно также определить другие формы знаний. Например, ниже приведено определение “знаний о погоде”.

$$\text{KnowsWhether}(a, p) \Leftrightarrow \text{Knows}(a, p) \vee \text{Knows}(a, \neg p)$$

Продолжая приведенный выше пример, отметим: Лойс знает, может ли Кларк летать, если она либо знает, что Кларк может летать, либо знает, что он этого не может.

Понятие “знаний о чем-то” является более сложным. Например, может возникнуть соблазн утверждать, что агент знает номер телефона Боба, если есть какой-то объект  $x$ , в отношении которого агент знает, что  $\text{PhoneNumber}(\text{Bob}) = x$ . Но этого недостаточно, поскольку агент может знать, что Алиса и Боб имеют один и тот же номер (например,  $\text{PhoneNumber}(\text{Bob}) = \text{PhoneNumber}(\text{Alice})$ ), но если номер Алисы неизвестен, то эти знания не могут ему помочь. Лучшее определение понятия “знаний о чем-то” состоит в том, что агент должен знать о некотором  $x$ , что это — строка цифр и что это — номер Боба:

$$\begin{aligned} \text{KnowsWhat}(a, \text{PhoneNumber}(b)) &\Leftrightarrow \\ \exists x \text{ Knows}(a, \underline{x} = \text{PhoneNumber}(\underline{b})) \wedge x \in \text{DigitStrings} \end{aligned}$$

Безусловно, применительно к другим вопросам необходимо иметь другие критерии определения того, какой ответ является приемлемым. Например, подходящим ответом на вопрос: “Какой город является столицей штата Нью-Йорк”, является имя собственное, “Олбани”, а не нечто подобное высказыванию: “Город, в котором находится резиденция губернатора штата”. Для обеспечения этого необходимо оформить предикат  $\text{KnowsWhat}$  в виде трехместного отношения: он принимает в качестве параметра обозначение агента, строку, представляющую некоторый терм, и категорию, к которой должен принадлежать ответ. Например, для определения упомянутых выше знаний могут использоваться следующие формулировки:

$$\begin{aligned} \text{KnowsWhat}(\text{Agent}, \text{Capital}(\text{NewYork}), \text{ProperNames}) \\ \text{KnowsWhat}(\text{Agent}, \text{PhoneNumber}(\text{Bob}), \text{DigitStrings}) \end{aligned}$$

### Знания, время и действия

В большинстве реальных ситуаций любому агенту приходится иметь дело с убеждениями (своими собственными или других агентов), которые изменяются во времени. Агенту также приходится составлять планы, касающиеся изменения своих собственных убеждений, такие как план покупки карты для определения способа попасть в Бухарест. Как и в случае других предикатов, мы можем овеществить предикат  $\text{Believes}$  и вести речь об убеждениях, которые имели место в течение определенного периода. Например, чтобы сформулировать высказывание “Лойс сегодня уверена в том, что Супермен может летать”, мы будем записывать следующее:

$$T(\text{Believes}(\text{Lois}, \text{Flies}(\text{Superman})), \text{Today})$$

Если объектом убеждения является высказывание, которое может изменяться во времени, и это можно описать, используя оператор  $T$  внутри строки. Например, Лойс может сегодня быть убеждена в том, что Супермен мог летать вчера:

$$T(\text{Believes}(\text{Lois}, T(\text{Flies}(\text{Superman}), \text{Yesterday})), \text{Today})$$

После получения способа описания того, как убеждения меняются со временем, мы приобретаем возможность использовать всю механику исчисления событий для составления планов, касающихся убеждений. Действия могут включать **предусловия знаний** и **результаты знаний**. Например, предусловием действия по набору номера телефона некоторого лица является знание этого номера, а результатом действия по

поиску номера становится знание этого номера. Например, последнее действие можно описать с использованием средств исчисления событий следующим образом:

```
Initiates(Lookup(a, "PhoneNumber(b) ")  
KnowsWhat(a, "PhoneNumber(b)", DigitStrings), t)
```

Планы сбора и использования информации часто представляют с помощью сокращенных обозначений, известных под названием **переменных этапа прогона** (runtime variable), соглашения по использованию которых тесно связаны с соглашениями о применении переменных, взятых без кавычек, которые были описаны выше. Например, план поиска номера Боба, а затем набора этого номера может быть записан следующим образом:

```
[Lookup(Agent, "PhoneNumber(Bob)", n), Dial(n)]
```

Здесь n — это переменная этапа прогона, с которой будет связано значение путем выполнения действия *Lookup*, поэтому данная переменная может затем использоваться в действии *Dial*. Планы такого рода часто встречаются в частично наблюдаемых проблемных областях. Примеры таких планов рассматриваются в следующем разделе и в главе 12.

## 10.5. МИР ПОКУПОК В INTERNET

В данном разделе показано, как можно представить некоторые знания, касающиеся осуществления покупок в Internet. Мы создадим агента по исследованию возможности осуществления покупок (торгового агента), который помогает покупателю найти в Internet предложения по интересующим его товарам. Торговый агент получает описание товара от покупателя и выполняет задачу по подготовке списка Web-страниц, на которых данный товар предлагается на продажу. В некоторых случаях описание покупателем товара будет точным, таким как “Цифровая камера Coolpix 995”, и тогда задача заключается в поиске магазина (магазинов) с наилучшим предложением. В других случаях описание может быть определено лишь частично, как, например, “Цифровая камера не дороже 300 долларов”, и агенту придется сравнивать различные товары.

Средой существования торгового агента является вся система World Wide Web — не игрушечная моделированная среда, а та сложная, постоянно развивающаяся среда, которая используется каждый день миллионами людей. Результатами восприятиями агента являются Web-страницы, но в отличие от человека — пользователя Web, который рассматривает страницы, отображенные в виде массива пикселов на экране, торговый агент воспринимает страницу в виде символьной строки, состоящей из обычных слов, чередующихся с командами форматирования на языке разметки HTML. В листинге 10.1 показаны Web-страница и соответствующая символьная строка HTML. Проблема восприятия для торгового агента состоит в извлечении полезной информации из восприятий такого рода.

Безусловно, что решить задачу восприятия Web-страниц проще по сравнению, скажем, с восприятием дорожной ситуации во время вождения такси в Каире. Тем не менее задача восприятия в Internet усложняется многими факторами. Web-страница, показанная в листинге 10.1, является очень простой по сравнению с реальными торговыми узлами, которые включают cookie-файлы, код Java, Javascript,

Flash, протоколы защиты от роботов, плохо сформированный код HTML, звуковые файлы, фильмы и текст, который появляется только в составе графического изображения, например в формате JPEG. Агент, способный действовать в пределах всей Internet, должен быть почти таким же сложным, как и робот, способный двигаться в реальном мире. Поэтому мы сосредоточимся на описании простого агента, в котором не учитывается большая часть этих осложнений.

**Листинг 10.1.** Web-страница типичного оперативного магазина в форме, воспринимаемой человеком — пользователем браузера (вверху), и соответствующая строка HTML, принимаемая браузером или воспринимаемая торговым агентом (внизу). В языке HTML символы между знаками < и > представляют собой директивы разметки, которые указывают, как должна отображаться страница. Например, строка `<i>Select</i>` означает, что нужно переключиться на курсивный шрифт, отобразить слово *Select*, а затем отменить использование курсивного шрифта. Идентификатор страницы, такой как `http://gen-store.com/books`, называется унифицированным локатором ресурсов, или URL (Uniform Resource Locator). Разметка `<a href="url">anchor</a>` означает, что должна быть создана гипертекстовая ссылка на локатор `url` с текстом анкера `anchor`

#### Generic Online Store

Select from our fine line of products:

- [Computers](#)
- [Cameras](#)
- [Books](#)
- [Videos](#)
- [Music](#)

```
<h1>Generic Online Store</h1>
<i>Select</i> from our fine line of products:
<ul>
<li> <a href="http://gen-store.com/compu">Computers</a>
<li> <a href="http://gen-store.com/camer">Cameras</a>
<li> <a href="http://gen-store.com/books">Books</a>
<li> <a href="http://gen-store.com/video">Videos</a>
<li> <a href="http://gen-store.com/music">Music</a>
</ul>
```

В первую очередь задача агента состоит в том, чтобы найти соответствующие товарные предложения (ниже будет показано, как выбрать наилучшие из искомых предложений). Допустим, что *query* — описание товара, которое вводит пользователь (например, “laptops” — портативные компьютеры); в таком случае некоторая страница представляет собой предложение, соответствующее запросу *query*, если страница действительно ему соответствует и содержит подходящее предложение. Мы должны также следить за URL, связанным с этой страницей:

$$\text{RelevantOffer}(\text{page}, \text{url}, \text{query}) \Leftrightarrow \text{Relevant}(\text{page}, \text{url}, \text{query}) \wedge \\ \text{Offer}(\text{page})$$

Запросу будет соответствовать страница с обзором новейших портативных компьютеров высокого класса, но если она не предоставляет способ купить такой компьютер, то ее нельзя рассматривать как предложение. Например, можно утверждать, что страница является предложением, если на ней содержится слово “buy” (покупка) или “price” (цена) в какой-то ссылке HTML или в форме, относящейся к этой странице. Иными словами, если страница содержит строку в виде “`<a ... buy ... </a> ...`”, то является предложением; на ней может также применяться слово “price” вместо “buy”, а вместо дескрип-

тора “*a*” использоваться дескриптор “*form*”. Аксиомы, относящиеся к этим утверждениям, могут быть записаны следующим образом:

$$\begin{aligned} Offer(page) &\Leftrightarrow (InTag("a", str, page) \vee InTag("form", str, page)) \\ &\quad \wedge (In("buy", str) \vee In("price", str)) \\ InTag(tag, str, page) &\Leftrightarrow In("<" + tag + str + "</" + tag, page) \\ In(sub, str) &\Leftrightarrow \exists i \ str[i:i+Length(sub)] = sub \end{aligned}$$

Теперь необходимо найти соответствующие страницы. Применимая для этого стратегия состоит в том, чтобы начать с начальной страницы некоторого оперативного магазина и рассмотреть все страницы, которых можно достичь, следуя по соответствующим ссылкам<sup>6</sup>. Агент должен обладать знаниями о многих магазинах, например, в таком виде:

$$\begin{aligned} Amazon \in OnlineStores \wedge Homepage(Amazon, "amazon.com") \\ Ebay \in OnlineStores \wedge Homepage(Ebay, "ebay.com") \\ GenStore \in OnlineStores \wedge Homepage(GenStore, "gen-store.com") \end{aligned}$$

В этих магазинах товары классифицируются по категориям товаров, а на начальной странице предоставляются ссылки на основные категории товаров. К младшим категориям можно перейти, следуя по цепочке соответствующих ссылок, что позволяет в конечном итоге проложить путь к требуемым предложениям. Иными словами, страница соответствует запросу, если ее можно достичь, проследовав по цепочке соответствующих ссылок на категорию от начальной страницы магазина, а затем пройдя еще по одной ссылке к предложению товара:

$$\begin{aligned} Relevant(page, url, query) \Leftrightarrow \\ \exists store, home \ store \in OnlineStores \wedge Homepage(store, home) \\ \wedge \exists url_2 \ RelevantChain(home, url_2, query) \wedge Link(url_2, url) \\ \wedge page = GetPage(url) \end{aligned}$$

Здесь предикат *Link*(*from*, *to*) означает, что имеется гиперссылка от URL *from* к URL *to* (см. упр. 10.13); чтобы определить, какая цепочка ссылок может рассматриваться как соответствующая запросу, *RelevantChain*, необходимо следовать не просто по любым существующим гиперссылкам, а только по таким ссылкам, связанный с которыми текст анкера указывает, что эта ссылка соответствует данному запросу на приобретение товара. Для этого используется предикат *LinkText*(*from*, *to*, *text*), который означает, что имеется ссылка между URL *from* и *to*, текстом анкера которой является *text*. Цепочка ссылок между двумя URL, *start* и *end*, соответствует описанию *d*, если текст анкера каждой ссылки представляет собой соответствующее имя категории для *d*. Существование самой цепочки определяется с помощью рекурсивной формулировки, в которой в качестве базового случая используется пустая цепочка (*start=end*):

$$\begin{aligned} RelevantChain(start, end, query) \Leftrightarrow (start=end) \vee \\ (\exists u, text \ LinkText(start, u, text) \wedge \\ RelevantCategoryName(query, text) \wedge \\ RelevantChain(u, end, query)) \end{aligned}$$


---

<sup>6</sup> Альтернативной стратегии, предусматривающей прохождение по ссылкам, является использование какой-то машины поиска в Internet; технология, лежащая в основе поиска в Internet и выборки информации, рассматривается в разделе 23.2.

Теперь необходимо определить, что подразумевается под высказыванием, будто текст *text* является соответствующим именем категории *RelevantCategoryName* для запроса *query*. Прежде всего необходимо связать строки с категориями, именуемыми с помощью этих строк. Такую задачу можно решить с использованием предиката *Name(s, c)*, который сообщает, что строка *s* является именем категории *c*, например, чтобы можно было сформулировать утверждение, что *Name("laptops", LaptopComputers)*. Некоторые дополнительные примеры применения предиката *Name* приведены в табл. 10.1, б. Затем определим понятие соответства. Допустим, что запросом *query* является "laptops" (Портативные компьютеры). В таком случае предикат *RelevantCategoryName(query, text)* принимает истинное значение, когда выполняется одно из приведенных ниже условий.

**Таблица 10.1. Примеры обозначений категорий**

Таксономия категорий товаров	Слова, которыми обозначаются эти категории
a)	б)
<i>Books ⊂ Products</i>	<i>Name("books", Books)</i>
<i>MusicRecordings ⊂ Products</i>	<i>Name("music", MusicRecordings)</i>
<i>MusicCDs ⊂ MusicRecordings</i>	<i>Name("CDs", MusicCDs)</i>
<i>MusicTapes ⊂ MusicRecordings</i>	<i>Name("tapes", MusicTapes)</i>
<i>Electronics ⊂ Products</i>	<i>Name("electronics", Electronics)</i>
<i>DigitalCameras ⊂ Electronics</i>	<i>Name("digital cameras", DigitalCameras)</i>
<i>StereoEquipment ⊂ Electronics</i>	<i>Name("stereos", Stereo Equipment)</i>
<i>Computers ⊂ Electronics</i>	<i>Name("computers", Computers)</i>
<i>LaptopComputers ⊂ Computers</i>	<i>Name("laptops", LaptopComputers)</i>
<i>DesktopComputers ⊂ Computers</i>	<i>Name("desktops", DesktopComputers)</i>
...	...

- Строки *text* и *query* именуют одну и ту же категорию, например "laptop computers" и "laptops".
- В строке *text* именуется надкатегория, такая как "computers".
- В строке *text* именуется подкатегория, такая как "ultralight notebooks".

Логическое определение предиката *RelevantCategoryName* состоит в следующем:

$$\begin{aligned} \text{RelevantCategoryName}(\text{query}, \text{text}) \Leftrightarrow \\ \exists c_1, c_2 \text{ Name } (\text{query}, c_1) \wedge \text{Name}(\text{text}, c_2) \\ \wedge (c_1 \subseteq c_2 \vee c_2 \subseteq c_1) \end{aligned} \quad (10.1)$$

В противном случае текст анкера не соответствует запросу, поскольку в нем именуется категория, не соответствующая данному описанию, такая как "mainframe computers" (мэйнфреймы) или "lawn & garden" (газон и сад).

Поэтому, чтобы обладать способностью следовать по искомым ссылкам, необходимо иметь широкие знания об иерархии категорий товаров. Верхняя часть этой иерархии может выглядеть, как показано в табл. 10.1, а. Задача перечисления всех возможных товарных категорий неосуществима, поскольку у покупателя всегда могут

возникнуть какие-то новые пожелания, а производители всегда готовы предложить новые товары для их удовлетворения (возможно, скоро будут даже продаваться обогреватели для коленных чашечек?). Тем не менее онтология, состоящая примерно из тысячи категорий, должна оказаться очень полезным инструментом для большинства покупателей.

Кроме самой иерархии товаров, необходимо иметь богатый словарь имен для категорий. Жизнь была бы гораздо проще, если бы существовало взаимно однозначное соответствие между категориями и символыми строками, которые их именуют. Мы уже сталкивались с проблемой **синонимов** — двух имен для одной и той же категории, таких как "laptop computers" and "laptops". Кроме того, существует также проблема **омонимов** — обозначения одним именем двух или нескольких разных категорий. Например, если в базу знаний, приведенную в табл. 10.1, б, будет добавлено следующее вполне допустимое высказывание, что словом "CDs" обозначаются депозитные сертификаты:

```
Name( "CDs" , CertificatesOfDeposit )
```

то имя "CDs" будет относиться к двум различным категориям.

Наличие синонимов и омонимов может стать причиной значительного увеличения количества путей, по которым должен следовать агент, и иногда может затруднить определение того, действительно ли данная страница соответствует запросу. Гораздо более серьезная проблема состоит в том, что пользователь может вводить очень широкий спектр описаний, а магазин может использовать не такие имена категорий, о которых знает агент. Например, в ссылке может быть указано "laptop", тогда как в базе знаний имеется только имя "laptops", или пользователь может ввести запрос: "компьютер, который помещается на откидном столике сидения экономического класса в самолете Boeing 737". Невозможно заранее перечислить все способы, с помощью которых могут присваиваться имена какой-то категории, поэтому в некоторых случаях агент должен проявлять способность проводить дополнительные рассуждения для определения того, соответствует ли рассматриваемый текст отношению *Name*. В наихудшем случае для этого требуется полное понимание естественного языка, но отложим эту тему до главы 22. На практике многое можно сделать с помощью нескольких простых правил, например, позволяющих согласовать слово "laptop" с категорией, названной как "laptops". В упр. 10.15 предлагается разработать множество таких правил после проведения некоторых исследований, касающихся оперативных магазинов.

Готовы ли мы применить некоторый алгоритм логического вывода для получения множества предложений, соответствующих нашему запросу, после того как составлены логические определения, описанные в предыдущих абзацах, и подготовлены приемлемые базы знаний о категориях товаров и соглашениях об именовании? К сожалению, не совсем! Недостающим элементом является функция *GetPage(url)*, которая ссылается на HTML-страницу, находящуюся по заданному URL. Агент не хранит в своей базе знаний содержимое страниц с каждым URL, а также не располагает явными правилами дедуктивного вывода того, каким должно быть это содержимое. Вместо этого можно предусмотреть, чтобы правильная процедура HTTP выполнялась каждый раз, когда какая-то подцель требует применения функции выборки страницы *GetPage*. Благодаря этому машина логического вывода сталкивается с такой ситуацией, как если бы вся система Web находилась в базе зна-

ний. В этом состоит пример применения общего метода, называемого **процедурным вложением**, с помощью которого конкретные предикаты и функции могут выполняться с использованием методов специального назначения.

### Сравнение коммерческих предложений

Теперь предположим, что в результате применения процессов формирования рассуждений, описанных в предыдущем разделе, подготовлено множество страниц с предложениями, касающимися нашего запроса "laptops". Чтобы сравнить эти предложения, агент должен извлечь из страниц с предложениями соответствующую информацию о цене, быстродействии, объеме диска, весе и т.д. Применительно к реальным Web-страницам эта задача может оказаться сложной, по всем тем же причинам, которые уже упоминались выше. Общепринятый способ решения этой проблемы состоит в использовании программ, называемых **оболочками** (wrapper), для извлечения информации из страницы. Технология извлечения информации будет описана в разделе 23.3. А пока предположим, что такие оболочки существуют и после получения доступа к странице и базе знаний они добавляют к базе знаний определенные утверждения. Как правило, для обработки страницы должна применяться определенная иерархия оболочек: наиболее общая оболочка извлекает информацию о датах и ценах, более конкретная извлекает данные об атрибутах товаров, относящихся к категории компьютеров, а в случае необходимости применяется оболочка, относящаяся к конкретному узлу, которая обладает сведениями о формате представления данных в конкретном магазине. Допустим, что имеется страница с узла `gen-store.com`, на которой находится текст:

YVM ThinkBook 970. Our price: \$1449.00

сопровождаемый данными о различных технических характеристиках. В этом случае желательно иметь оболочку для извлечения примерно такой информации:

```

$$\exists lc, offer \in LaptopComputers \wedge offer \in ProductOffers \wedge
  ScreenSize(lc, Inches(14)) \wedge ScreenType(lc, ColorLCD) \wedge
  MemorySize(lc, Megabytes(512)) \wedge CPUSpeed(lc, GHz(2.4)) \wedge
  OfferedProduct(offer, lc) \wedge Store(offer, GenStore) \wedge
  URL(offer, "genstore.com/comps/34356.html") \wedge
  Price(offer, $(449)) \wedge Date(offer, Today)$$

```

Этот пример иллюстрирует некоторые проблемы, которые возникают с самого начала серьезных исследований в области инженерии знаний для коммерческих сделок. Например, обратите внимание на то, что цена является атрибутом предложения *offer*, а не самого товара. Это важно, поскольку предложение в каждом конкретном магазине может изменяться со дня на день, даже на один и тот же портативный компьютер, а для некоторых категорий (такие как дома и картины) предложения по приобретению одного и того же отдельного объекта могут даже сопровождаться другими одновременными предложениями по покупке сопутствующих товаров по различным ценам. Возникают и многие другие осложнения, о которых еще не шла речь, например, возможность того, что цена будет зависеть от метода платежа и от прав покупателя на некоторые скидки. Так или иначе, но в этой области еще предстоит выполнить очень много интересной работы.

Последняя задача состоит в сравнении информации о предложениях, которая была извлечена из разных страниц. Например, рассмотрим следующие три предложения:

- A: 2.4 GHz CPU, 512MB RAM, 80 GB disk, DVD, CDRW, \$1695  
B: 2.0 GHz CPU, 1GB RAM, 120 GB disk, DVD, CDRW, \$1800  
C: 2.2 GHz CPU, 512MB RAM, 80 GB disk, DVD, CDRW, \$1800

Предложение A **доминирует** над предложением C; это означает, что предлагаемый на узле A компьютер дешевле и имеет более высокое быстродействие, а во всем остальном два компьютера одинаковы. Вообще говоря, предложение X доминирует над Y, если X имеет лучшее значение по меньшей мере одного атрибута и не хуже по всем остальным атрибутам. Но ни A, ни B не доминируют друг над другом. Чтобы определить, какое из этих предложений лучше, необходимо знать, как покупатель оценивает быстродействие процессора и стоимость по отношению к объему памяти и диска. Тема выбора предпочтений среди многочисленных атрибутов в целом рассматривается в разделе 16.4, а пока наш торговый агент просто возвращает список всех предложений, соответствующих описанию покупателя, над которыми не доминируют другие предложения. В данном примере ни одно предложение не доминирует ни над A, ни над B. Обратите внимание на то, что данный результат основан на предположении, что все предпочитают более низкие цены, более быстродействующие процессоры и больший объем памяти. Некоторые атрибуты, такие как размер экрана в ноутбуке, зависят от конкретных предпочтений пользователя (хочет ли он иметь большой экран или миниатюрный ноутбук); для решения вопроса о таких предпочтениях торговый агент должен получить ответ от пользователя.

Торговый агент, описанный в этом разделе, является весьма простым; для него возможны многие усовершенствования. Тем не менее этот агент обладает достаточными способностями, чтобы при наличии правильных знаний о конкретной проблемной области он мог действительно оказать помощь покупателю. Поскольку этот агент имеет декларативную конструкцию, его можно легко дополнить для использования в более сложных приложениях. Основной замысел данного раздела состоял в том, чтобы показать, что для агента, подобного этому, требуется определенное представление знаний (в частности, об иерархии товаров), а после приобретения некоторых знаний в такой форме уже не очень сложно создать все остальное, что требуется от агента, основанного на знаниях.

## 10.6. СИСТЕМЫ ФОРМИРОВАНИЯ РАССУЖДЕНИЙ О КАТЕГОРИЯХ

Выше было показано, что категории являются основными строительными блоками любой крупномасштабной схемы представления знаний. А в данном разделе описаны системы, специально предназначенные для организации и проведения рассуждений, касающихся категорий. Существуют два тесно связанных семейства систем: **семантические сети** предоставляют графические средства, позволяющие визуально представить базу знаний, и эффективные алгоритмы для логического вывода сведений о любом объекте на основании его принадлежности к некоторой категории, а **описательные логики** предоставляют формальный язык для конструирования и

комбинирования определений категорий, а также эффективные алгоритмы для установления связей между категориями на уровне подмножеств и надмножеств.

## Семантические сети

В 1909 году Чарльз Пирс предложил графическую систему обозначений в виде узлов и дуг, получившую название **экзистенциальных графов**, которую он назвал “логикой будущего”. Так зародилось понятие семантических сетей. С того времени и начались продолжительные дебаты между приверженцами “чистой логики” и приверженцами “чисто семантических сетей”. К сожалению, эти дебаты затмили тот факт, что семантические сети (по меньшей мере, те из них, в которых правильно определена семантика) представляют собой всего лишь одну из форм логики. Система обозначений для высказываний некоторых типов, предусмотренная в семантических сетях, часто является более удобной для человека, но если отбросить эти особенности, касающиеся “удобного интерфейса”, то базовые понятия (объекты, отношения, кванторы и т.д.) окажутся теми же самыми.

Существует много вариантов семантических сетей, но все они способны представлять отдельные объекты, категории объектов и отношения между объектами. В типичной графической системе обозначений имена объектов или категорий изображаются в овалах или прямоугольниках, а связи между ними обозначаются с помощью дуг с метками. Например, на рис. 10.7 показана связь *MemberOf* между *Mary* и *FemalePersons*, которая соответствует логическому утверждению  $Mary \in FemalePersons$ ; аналогичным образом, связь *SisterOf* между *Mary* и *John* соответствует утверждению  $SisterOf(Mary, John)$ . Категории можно соединять с помощью связи *SubsetOf* и т.д. Рисовать эти овалы и стрелки так интересно, что часто можно забыть обо всем. Например, мы знаем, что материю людьми являются особы женского пола, но можно ли нарисовать связь *HasMother* от категории *Persons* к категории *FemalePersons*? Ответ является отрицательным, поскольку *HasMother* — это связь между человеком и его матерью, а категории не имеют матерей<sup>7</sup>. По этой причине на рис. 10.7 используется специальное обозначение — связь с меткой в прямоугольнике с двойным контуром. Эта связь соответствует следующему утверждению:

$$\forall x \ x \in Persons \Rightarrow [\forall y \ HasMother(x, y) \Rightarrow y \in FemalePersons]$$

Нам может также потребоваться сформулировать утверждение, что люди имеют две ноги, таким образом:

$$\forall x \ x \in Persons \Rightarrow Legs(x, 2)$$

Как и прежде, необходимо тщательно следить за тем, чтобы в этом высказывании не утверждалось, что ногами обладает категория; как показано на рис. 10.7, для

<sup>7</sup> В некоторых ранних системах не была предусмотрена возможность проводить различие между свойствами элементов категории и свойствами всей категории в целом. Это могло приводить непосредственно ко многим несогласованностям, как указал Дрю Макдермотт в своей статье “Artificial Intelligence Meets Natural Stupidity” (Искусственный интеллект сталкивается с естественной глупостью) [1020]. Еще одной распространенной проблемой было использование связей *IsA* и для отношений между подмножествами и множествами, и для отношений между элементами и подмножествами, в полном соответствии с грамматикой английского языка: “a cat is a mammal” (Кошка — млекопитающее) и “Fifi is a cat” (Фифи — кошка). Дополнительная информация по этой теме приведена в упр. 10.25.

формулировки утверждений о свойствах каждого элемента категории используется связь, обозначенная меткой в прямоугольнике с одинарным контуром.

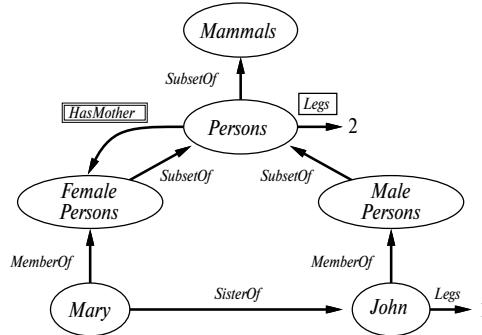


Рис. 10.7. Семантическая сеть с четырьмя объектами (John, Mary, 1 и 2) и четырьмя категориями. Отношения обозначаются связями с метками

Система обозначений с использованием семантических сетей обеспечивает значительные удобства при формировании рассуждений с учетом **наследования** такого типа, которые были впервые представлены в разделе 10.2. Например, благодаря тому, что Мэри является человеком, она наследует свойство иметь две ноги. Поэтому, чтобы узнать, сколько ног имеет Мэри, алгоритм наследования проходит по связи *MemberOf* от объекта *Mary* к категории, к которой она принадлежит, а затем проходит по связям *SubsetOf* вверх по иерархии до тех пор, пока не находит категорию, для которой имеется связь с обведенной прямоугольником меткой *Legs*, в данном случае категории *Persons*. Простота и эффективность этого механизма логического вывода по сравнению с логическим доказательством теорем издавна была одной из основных привлекательных особенностей семантических сетей.

Наследование становится сложнее, если некоторый объект может принадлежать больше чем к одной категории или если категория может быть подмножеством больше чем одной отличной от нее категории; такая ситуация называется **множественным наследованием**. В подобных случаях алгоритм наследования может находить два или несколько конфликтующих значений, представляющих собой ответ на запрос. По этой причине множественное наследование запрещено в некоторых языках **объектно-ориентированного программирования** (Object-Oriented Programming — ООП), таких как язык Java, в котором используется наследование в иерархии классов. В семантических сетях множественное наследование обычно разрешено, но мы отложим обсуждение этой темы до раздела 10.7.

Еще одной общей формой логического вывода является использование **инверсных связей**. Например, связь *HasSister* является инверсной по отношению к *SisterOf*, а это означает, что имеет место следующее высказывание:

$$\forall p, s \text{ HasSister}(p, s) \Leftrightarrow \text{SisterOf}(s, p)$$

Это высказывание может быть подтверждено в семантической сети, если связи являются **овеществленными**, т.е. преобразованными в объекты в полном смысле этого понятия. Например, может быть предусмотрен объект *SisterOf*, соединен-

ный связью *Inverse* с объектом *HasSister*. После получения запроса, касающегося того, кто является сестрой (*SisterOf*) Джона, алгоритм логического вывода может обнаружить, что связь *HasSister* является инверсной по отношению к связи *SisterOf*, и поэтому ответить на данный запрос, проследовав по связи *HasSister* от объекта *John* к объекту *Mary*. При отсутствии этой инверсной информации могло бы потребоваться проверить каждое лицо женского пола для определения того, не имеет ли это лицо связь *SisterOf*, направленную к Джону. Это связано с тем, что в семантических сетях непосредственная индексация предусмотрена только для самих объектов, а также категорий и связей, исходящих из них; проводя сравнение с логикой первого порядка, можно отметить, что это аналогично индексации базы знаний только по первому параметру каждого предиката.

Читатель должен был обнаружить очевидный недостаток системы обозначений с помощью семантических сетей по сравнению с логикой первого порядка: тот факт, что между овалами проводятся связи, соединяющие два овала, говорит о том, что в семантических сетях могут быть представлены только бинарные отношения. Например, высказывание *Fly(Shankar, NewYork, NewDelhi, Yesterday)* не может быть реализовано непосредственно в семантической сети. Тем не менее можно достичь эффекта реализации *n*-арных высказываний путем овеществления самого рассматриваемого высказывания в виде события (см. раздел 10.3), принадлежащего к соответствующей категории событий. На рис. 10.8 показана структура семантической сети для указанного выше конкретного события. Обратите внимание на то, что ограничение, связанное с необходимостью применять только бинарные отношения, вынуждает создавать богатую онтологию овеществленных понятий; в действительности основная часть онтологии, разработанной в данной главе, происходила из систем с семантическими сетями.

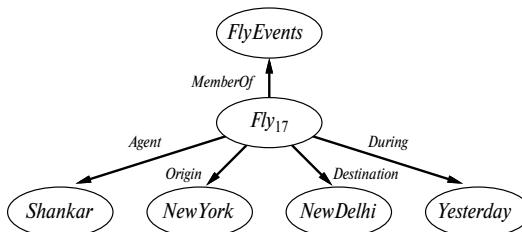


Рис. 10.8. Фрагмент семантической сети, на которой показано представление логического утверждения *Fly(Shankar, NewYork, NewDelhi, Yesterday)*

Овеществление высказываний дает возможность представить в системе обозначений семантической сети любое базовое, атомарное высказывание логики первого порядка, не содержащее функций. Реализация некоторых видов высказываний с кванторами всеобщности может осуществляться с помощью инверсных связей и применяемых к категориям стрелок с метками в одинарных и двойных прямоугольниках, но даже такие ухищрения отнюдь не позволяют приблизиться к возможностям полной логики первого порядка. В этой системе обозначений недостает всех таких выразительных возможностей, как отрицание, дизъюнкция, вложенные функциональные символы и кванторы существования. В последнее время появилась возможность расширить эту систему обозначений, чтобы сделать ее эквивалентной

логике первого порядка (как при использовании экзистенциальных графов Пирса или секционированных семантических сетей Хендрикса [647]), но это приводит к исчезновению одного из основных преимуществ семантических сетей — простоты и прозрачности процессов логического вывода. Простая система обозначений позволяет проектировщикам создавать большие сети и все еще иметь полное представление о том, какие запросы должны быть эффективными, поскольку, во-первых, можно легко визуально представить, через какие шаги должна пройти процедура логического вывода, и, во-вторых, в некоторых случаях язык запросов настолько прост, что в нем невозможно сформулировать сложные, запутанные запросы. Для тех случаев, когда обнаруживается, что выразительная мощь слишком ограничена, во многих системах семантических сетей предусмотрены **процедурные вложения**, позволяющие компенсировать эти недостатки. Процедурное вложение — это метод, с помощью которого при выполнении запроса, касающегося некоторого отношения (а иногда и при вводе в базу знаний некоторого утверждения), осуществляется вызов специальной процедуры (предназначенной лишь для обработки этого отношения), а не общего алгоритма логического вывода.

Одной из самых важных характерных особенностей семантических сетей является их способность представлять **заданные по умолчанию значения** для категорий. Внимательно изучив рис. 10.7, можно обнаружить, что Джон имеет одну ногу, несмотря на тот факт, что он — человек, а все люди имеют две ноги. В строго логической базе знаний такое сочетание фактов было бы противоречивым, но в семантической сети утверждение, что все люди имеют две ноги, обладает лишь статусом значения, применяемого по умолчанию; т.е. применительно к любому человеку предполагается, что он имеет две ноги, если этому не противоречит более конкретная информация. Принудительное применение семантики заданных по умолчанию значений осуществляется естественным образом с помощью алгоритма наследования, поскольку этот алгоритм следует по связям, направленным вверх от самого объекта (в данном случае Джона), и останавливается, как только находит искомое значение. Принято говорить, что заданное по умолчанию значение **перекрывается** более конкретным значением. Обратите внимание на то, что можно было бы также перекрыть информацию о заданном по умолчанию количестве ног, создав категорию одноногих людей *OneLeggedPersons*, подмножество категории *Persons*, элементом которого является *John*.

Можно было бы также сохранить строго логическую семантику для этой сети, сформировав следующее высказывание, что утверждение *Legs* для *Persons* содержит исключение для *John*:

$$\forall x \ x \in \text{Persons} \wedge x \neq \text{John} \Rightarrow \text{Legs}(x, 2)$$

При наличии фиксированной сети такое дополнение было бы семантически адекватным, но стало бы гораздо менее кратким по сравнению с самой сетевой системой обозначений, если бы и без этого в сети было бы много исключений. А в сети, которая обновляется путем добавления дополнительных утверждений, такой подход оказался бы полностью неприемлемым, поскольку фактически потребовалось бы указывать, что к исключениям относятся все люди, в отношении которых еще не известно, имеют ли они две или только одну ногу. В разделе 10.7 приведены дополнительные сведения об этой проблеме и о формировании рассуждений в логике умолчаний в целом.

## Описательные логики

Синтаксис логики первого порядка предназначен для упрощения процедуры формирования высказываний об объектах, а **описательные логики** представляют собой системы обозначений, которые предназначены для упрощения процедуры описания определений и свойств категорий. Системы описательной логики развились из семантических сетей в ответ на необходимость формализовать тот смысл, который несет в себе сеть, сохранив вместе с тем акцент на использование таксономической структуры в качестве принципа организации.

Основные задачи логического вывода для описательных логик сводятся к **обобщению** (проверке того, является ли одна категория подмножеством другой путем сравнения их определений) и **классификации** (определению принадлежности некоторого объекта к какой-то категории). В некоторых системах предусматривается также проверка **непротиворечивости** категории, т.е. того, являются ли выполнимыми критерии принадлежности к категории с точки зрения логики.

Типичным языком описательной логики является Classic [155]. Синтаксис описаний<sup>8</sup> Classic показан в листинге 10.2. Например, чтобы сформулировать утверждение, что холостяками называют неженатых взрослых мужчин, можно записать следующее:

```
Bachelor = And(Unmarried, Adult, Male)
```

Эквивалент этого утверждения в логике первого порядка выглядел бы так:

```
Bachelor(x) ⇔ Unmarried(x) ∧ Adult(x) ∧ Male(x)
```

### Листинг 10.2. Синтаксис описаний в подмножестве языка Classic

---

```

Concept → Thing | ConceptName
| And(Concept, ...)
| All(RoleName, Concept)
| AtLeast(Integer, RoleName)
| AtMost(Integer, RoleName)
| Fills(RoleName, IndividualName, ...)
| SameAs(Path, Path)
| OneOf(IndividualName, ...)
Path → [RoleName, ...]

```

---

Следует отметить, что в этом языке описательной логики фактически разрешается непосредственно выполнять логические операции с предикатами, что исключает необходимость в первую очередь создавать высказывания, которые должны быть соединены связками. Любое описание на языке Classic может быть сформулировано и в логике первого порядка, но некоторые описания Classic формулируются проще. Например, чтобы описать множество мужчин, имеющих трех сыновей, из которых все безработны и женаты на врачах, и, самое большое, двух дочерей, из которых все являются преподавателями на кафедрах физики или математики, можно записать следующее:

---

<sup>8</sup> Следует отметить, что этот язык не позволяет просто формулировать утверждение, что одно понятие (или категория) является подмножеством другого. Это требование введено сознательно: отношение обобщения между категориями должно быть выведено логическим путем из некоторых аспектов описаний категорий. Если этого не удается сделать, то в описаниях имеются какие-то упущения.

```
And(Man, AtLeast(3, Son), AtMost(2, Daughter),
    All(Son, And(Unemployed, Married, All(Spouse, Doctor))),
    All(Daughter, And(Professor, Fills(Department, Physics, Math))))
```

Перевод этого высказывания на язык логики первого порядка оставляем читателю в качестве упражнения.

По-видимому, одной из наиболее важных характерных особенностей описательных логик является сделанный в них акцент на осуществимости логического вывода. Решение любого экземпляра проблемы осуществляется путем его описания, а затем выполнения запроса, касающегося того, является ли этот экземпляр обобщением одной из нескольких возможных категорий решений. В стандартных системах логики первого порядка предсказание времени выработки решения часто оказывается невозможным, а пользователю чаще всего самому приходится разрабатывать представление, позволяющее исключать множества высказываний, которые, по-видимому, вынудят систему выполнять вычисления в течение нескольких недель, чтобы решить задачу. С другой стороны, в описательных логиках все направлено на обеспечение того, чтобы проблема проверки обобщения могла быть решена за время, полиномиально зависящее от размера описаний<sup>9</sup>.

На первый взгляд такое свойство описательных логик может показаться удивительным, пока не станет очевидно, что в процессе формулировки задачи может быть достигнут только один из двух безуспешных результатов: либо задача окажется настолько сложной, что ее описание вообще невозможно будет сформулировать, либо она потребует экспоненциально большого описания! Тем не менее анализ осуществимости логического вывода позволяет пролить свет на то, какого рода конструкции вызывают проблемы, и поэтому помочь пользователю понять, какие следствия вытекают из использования различных представлений. Например, в описательных логиках обычно не используются такие отношения, как отрицание и дизъюнкция. Дело в том, что каждое из этих отношений вынуждает логические системы первого порядка для обеспечения полноты проходить через этап анализа вариантов, который может потенциально характеризоваться экспоненциальной сложностью. По той же причине эти отношения исключены из языка Prolog. В языке Classic допускается использовать только ограниченную форму дизъюнкции в конструкциях *Fills* и *OneOf*, которые допускают выполнение дизъюнкции по явно заданным объектам, а не по их описаниям. Если бы было разрешено использовать дизъюнктивные описания, то вложенные определения могли бы легко привести к появлению экспоненциального количества альтернативных путей, по которым одна категория могла бы обобщать другую.

## 10.7. ФОРМИРОВАНИЕ РАССУЖДЕНИЙ С ИСПОЛЬЗОВАНИЕМ ИНФОРМАЦИИ, ЗАДАННОЙ ПО УМОЛЧАНИЮ

В предыдущем разделе был приведен простой пример утверждения с заданной по умолчанию информацией о состоянии: люди имеют две ноги. Это заданное по умолчанию значение можно переопределить с помощью более конкретной информации,

<sup>9</sup> В принципе, язык Classic обеспечивает эффективную проверку обобщения, но в худшем случае времени прогона может стать экспоненциальным.

например, такой, что Длинный Джон Сильвер имеет одну ногу. Кроме того, было показано, что механизм наследования в семантических сетях предоставляет простой и естественный способ переопределения значений, заданных по умолчанию. В этом разделе проводится более общее исследование заданной по умолчанию информации, с тем чтобы можно было понять семантику умолчаний, а не просто создать какой-то процедурный механизм.

### Открытые и закрытые миры

Предположим, что вы просматриваете доску объявлений университетской кафедры компьютерных наук и видите сообщение со словами: “Студентам будут предложены следующие курсы лекций: CS 101, CS 102, CS 106, EE 101”. Итак, сколько курсов здесь предлагается? Если вы ответите “четыре”, то поступите в соответствии с принципами работы типичной системы баз данных. После ввода информации в реляционную базу данных с помощью оператора, эквивалентного следующим четырем утверждениям:

$$\begin{aligned} \text{Course } (\text{CS}, 101), \quad \text{Course } (\text{CS}, 102), \quad \text{Course } (\text{CS}, 106), \\ \text{Course } (\text{EE}, 101) \end{aligned} \tag{10.2}$$

запрос на языке SQL `select count(*) from Course` возвратит 4. С другой стороны, логическая система первого порядка, скорее всего, ответит примерно в том смысле, что количество предлагаемых курсов должно находиться в пределах от одного до бесконечности, но только не быть равно “четырем”. Причина этого состоит в том, что эти утверждения *Course* не отрицают возможности, что могут также предлагаться другие не упомянутые здесь курсы, а также не позволяют сделать вывод, что все указанные курсы отличаются друг от друга.

Этот пример показывает, что системы баз данных и соглашения, неявно применяемые людьми при общении, отличаются от логики первого порядка по меньшей мере в двух аспектах. Во-первых, в языках баз данных (и в языках общения людей) предполагается, что предоставленная информация является полной, поэтому предполагается, что базовые атомарные высказывания, не обозначенные как истинные, являются ложными. Такое предположение называется **предположением о замкнутом мире**, или CWA (Closed-World Assumption). Во-вторых, обычно предполагается, что разные имена относятся к различным объектам. В этом состоит **предположение об уникальности имен**, или UNA (Unique Names Assumption), которое было впервые описано в контексте имен действий в разделе 10.3.

В логике первого порядка такие соглашения не приняты, и поэтому необходимо их определять явно. Чтобы сформулировать высказывание, что предлагаются четыре и только четыре разных курса, необходимо записать следующее:

$$\begin{aligned} \text{Course}(d, n) \Leftrightarrow [d, n] = [\text{CS}, 101] \vee [d, n] = [\text{CS}, 102] \\ \vee [d, n] = [\text{CS}, 106] \vee [d, n] = [\text{EE}, 101] \end{aligned} \tag{10.3}$$

Уравнение 10.3 называется **дополнением**<sup>10</sup> уравнения 10.2. Вообще говоря, такое дополнение должно содержать определение для каждого предиката (высказывание в форме “тогда и только тогда”), а каждое определение должно содержать по одному

---

<sup>10</sup> Иногда это дополнение называют “дополнением Кларка” в честь предложившего его Кейта Кларка.

дизъюнкту для каждого определенного выражения, головой которого является этот предикат<sup>11</sup>. Как правило, такое дополнение создается, как описано ниже.

1. Собрать все выражения с одним и тем же названием предиката ( $P$ ) и с одной и той же арностью ( $n$ ).
2. Преобразовать каждое выражение в **нормальную форму Кларка**, например, заменить выражение

$$P(t_1, \dots, t_n) \leftarrow Body$$

где  $t_i$  — термины, выражением

$$P(v_1, \dots, v_n) \leftarrow \exists w_1 \dots w_m [v_1, \dots, v_n] = [t_1, \dots, t_n] \wedge Body$$

где  $v_i$  — вновь введенные переменные, а  $w_i$  — переменные, которые присутствуют в первоначальном выражении. При этом следует использовать одно и то же множество  $v_i$  для каждого выражения. Это приводит к получению такого множества выражений:

$$P(v_1, j, v_n) \leftarrow B_1$$

...

$$P(v_1, j, v_n) \leftarrow B_k$$

3. Скомбинировать их вместе в одно большое дизъюнктивное выражение:

$$P(v_1, \dots, v_n) \leftarrow B_1 \vee \dots \vee B_k$$

4. Сформировать завершение, заменив оператор  $\leftarrow$  оператором эквивалентности:

$$P(v_1, \dots, v_n) \Leftrightarrow B_1 \vee \dots \vee B_k$$

Пример дополнения Кларка для базы знаний, включающей и базовые факты, и правила, приведен в табл. 10.2. Чтобы ввести в действие предположение об уникальности имен, достаточно сформировать дополнение Кларка для отношения равенства, где единственными известными фактами являются  $CS=CS$ ,  $101=101$  и т.д. Оставляем эту задачу читателю в качестве упражнения.

**Таблица 10.2. Дополнение Кларка для множества хорновских выражений.** В первоначальной хорновской программе (слева) четыре курса перечислены явно, а также содержатся утверждения, что проводятся курсы по математике с целочисленными номерами от 101 до 130, а для каждого курса  $CS$  по компьютерным наукам в серии 100 (для студентов) имеется соответствующий курс в серии 200 (для аспирантов). В дополнении Кларка (справа) указано, что других курсов больше нет. С помощью этого дополнения и предположения об уникальности имен (а также очевидного определения предиката *Integer*) можно получить желаемое заключение, что имеется точно 36 курсов: 30 курсов по математике и 6 курсов по компьютерным наукам

Хорновские выражения	Дополнение Кларка
$Course(CS, 101)$	$Course(d, n) \Leftrightarrow [d, n] = [CS, 101]$
$Course(CS, 102)$	$\vee [d, n] = [CS, 102]$
$Course(CS, 106)$	$\vee [d, n] = [CS, 106]$
$Course(EE, 101)$	$\vee [d, n] = [EE, 101]$
$Course(EE, i) \leftarrow Integer(i)$	$\vee \exists i [d, n] = [EE, i] \wedge Integer(i)$

<sup>11</sup> Следует отметить, что эти выражения также представляют собой одну из разновидностей аксиом состояния-преемника, приведенных в разделе 10.3.

Окончание табл. 10.2

Хорновские выражения	Дополнение Кларка
$\wedge 101 \leq i \wedge i \leq 130$	$\wedge 101 \leq i \wedge i \leq 130$
$Course(CS, m+100) \Leftarrow$	$\vee \exists m [d, n] = [CS, m+100]$
$Course(CS, m) \wedge 100 \leq m$	$\wedge Course(CS, m) \wedge 100 \leq m$
$\wedge m < 200$	$\wedge m < 200$

Предположение о замкнутом мире позволяет найти **минимальную модель** отношения. Это означает, что в данном примере можно найти модель отношения *Course* с наименьшим количеством элементов. Минимальная модель отношения *Course*, соответствующая уравнению 10.2, имеет четыре элемента; при меньшем количестве элементов возникло бы противоречие. Для хорновских баз знаний всегда имеется уникальная минимальная модель. Следует отметить, что с учетом предположения об уникальности имен это утверждение распространяется также и на отношение равенства, поскольку каждый терм равен только самому себе. Как ни парадоксально, это означает также, что минимальные модели являются одновременно и максимальными, в том смысле, что включают максимально возможное количество объектов.

Было бы допустимо взять любую хорновскую программу, сформировать дополнение Кларка и передать полученный результат в программу автоматического доказательства теорем для выполнения логического вывода. Но обычно является более эффективным использование механизма логического вывода специального назначения, такого как Prolog, поскольку в этом механизме логического вывода уже учитываются предположения о замкнутом мире и уникальности имен.

Те, кому приходится использовать предположение о замкнутом мире, должны соблюдать осторожность при выборе способа формируемых рассуждений. Например, при использовании базы данных с результатами переписи населения было бы резонно принимать предположение о замкнутом мире при формировании рассуждений о текущем населении городов, но было бы неправильно делать вывод, что в будущем не родится ни один ребенок, лишь на основании того, что база данных не содержит записей, касающихся будущих дней рождения. Благодаря CWA база данных становится **полной** в том смысле, что позволяет дать либо положительный, либо отрицательный ответ на каждый атомарный запрос; если же какие-то факты не заданы в этой базе данных изначально (например, о будущих рождениях), то CWA не может применяться. Более сложная система представления знаний может дать пользователю возможность указывать правила, по которым следует применять предположение о замкнутом мире.

### Отрицание как недостижение цели и устойчивая семантика модели

Как было показано в главах 7 и 9, базы знаний в форме хорновских выражений имеют желаемые вычислительные свойства. Однако во многих приложениях довольно сложно выполнить выдвигаемое при этом требование, чтобы каждый литерал в теле выражения был положительным. Например, может потребоваться сформулировать утверждение: “На прогулку можно выходить, если нет дождя”, не будучи вынужденным изобретать такие предикаты, как *NotRaining*. В этом разделе рассматривается дополнение к хорновским выражениям в форме явного отрицания, которое

основано на идеи использования **отрицания как недостижения цели**. Эта идея состоит в том, что истинность отрицательного литерала  $\text{not } P$  может быть “доказана” просто на основании того, что попытка доказательства  $P$  окончилась неудачей. В этом заключается одна из форм рассуждений по умолчанию, тесно связанная с предположением о замкнутом мире: предполагается, что нечто является ложным, если нельзя доказать, что оно истинно. Для того чтобы можно было отличить отрицание как недостижение цели от логического оператора “ $\neg$ ”, для обозначения первого используется оператор “ $\text{not}$ ”.

В языке Prolog допускается применять оператор  $\text{not}$  в теле выражения. Например, рассмотрим следующую программу Prolog:

```
IDEdrive ← Drive ∧ not SCSIdrive.
SCSIdrive ← Drive ∧ not IDEdrive.
SCSIconroller ← SCSIdrive.
Drive. (10.4)
```

Первое правило указывает, что если в компьютере имеется жесткий диск, не являющийся диском SCSI, то это должен быть диск IDE. Согласно второму правилу, если это — не диск IDE, то он должен быть диском SCSI. В третьем правиле утверждается, что из наличия в компьютере диска SCSI следует наличие в нем контроллера SCSI, а четвертое утверждает, что в компьютере обязательно должен быть жесткий диск. Эта программа имеет следующие две минимальные модели:

```
M1 = {Drive, IDEdrive}
M2 = {Drive, SCSIdrive, SCSIconroller}
```

Очевидно, что минимальные модели не позволяют выразить намеченную семантику программ с отрицанием как недостижением цели. Рассмотрим следующую программу:

```
P ← not Q. (10.5)
```

Эта программа имеет две минимальные модели,  $\{P\}$  и  $\{Q\}$ . С точки зрения логики первого порядка в этом есть смысл, поскольку выражение  $P \leftarrow \neg Q$  эквивалентно  $P \vee Q$ . Но с точки зрения языка Prolog это сомнительно: если терм  $Q$  ни разу не появляется в этой программе слева от стрелки, то как же он может стать следствием?

Альтернативный подход состоит в использовании идеи **стабильной модели**, которая представляет собой такую минимальную модель, что каждый атом в модели имеет **обоснование**: правило, голова которого является атомом, а каждый литерал в теле выполняется. Формально модель  $M$  определяется как стабильная модель программы  $H$ , если  $M$  — уникальная минимальная модель **сокращения**  $H$  по отношению к  $M$ . Сокращение программы  $H$  определяется путем предварительного удаления из  $H$  любого правила, которое имеет в теле литерал  $\text{not } A$ , где  $A$  имеется в модели, а затем удаления всех отрицательных литералов в оставшихся правилах. Поскольку после этого сокращение программы  $H$  становится списком хорновских выражений, оно должно иметь уникальную минимальную модель.

Сокращением программы  $P \leftarrow \text{not } Q$  по отношению к  $\{P\}$  является  $P$ , и это сокращение имеет минимальную модель  $\{P\}$ . Поэтому  $\{P\}$  — стабильная модель. Сокращение по отношению к  $\{Q\}$  представляет собой пустую программу, и это сокращение имеет минимальную модель  $\{\}$ . Поэтому  $\{Q\}$  — это нестабильная модель,

поскольку  $Q$  не имеет обоснования в уравнении 10.5. В качестве еще одного примера можно привести сокращение уравнения 10.4 по отношению к  $M_1$ , как показано ниже.

```
IDEDdrive ← Drive.  

SCSIController ← SCSIDrive.  

Drive.
```

Эта программа имеет минимальную модель  $M_1$ , поэтому  $M_1$  — стабильная модель. ☐ **Программирование множества ответов** — это разновидность логического программирования с отрицанием как недостижением цели, которая функционирует на основе преобразования логической программы в базовую форму с последующим поиском стабильных моделей (называемых также ☐ **множествами ответов**) с использованием методов проверки пропозициональной модели. Таким образом, программирование множества ответов разработано на основе и системы Prolog, и быстродействующих алгоритмов автоматического доказательства выполнимости пропозициональных высказываний, таких как WalkSAT. И действительно, программирование множества ответов было успешно применено для решения задач планирования, по такому же принципу, как применялись программы автоматического доказательства выполнимости пропозициональных высказываний. Преимуществом планировщиков на основе множества ответов над другими планировщиками является их большая гибкость: операторы и ограничения планирования могут быть выражены в виде логических программ и не привязаны к жесткому формату конкретной формальной структуры планирования. Недостаток методов планирования на основе множества ответов является таким же, как и в других пропозициональных методах: если количество объектов в универсуме очень велико, то может возникать экспоненциальное замедление.

## Логика косвенного описания и логика умолчания

Выше приводились два примера, в которых с виду естественные процессы формирования рассуждений нарушают свойство **монотонности** логики, которое было доказано<sup>12</sup> в главе 7. В первом примере свойство, наследуемое всеми элементами категории в семантической сети, может быть переопределено с использованием более конкретной информации, касающейся подкатегории. Во втором примере отрицаемые литералы, выведенные на основании предположения о замкнутом мире, могут быть переопределены путем добавления положительных литералов.

Простой самоанализ показывает, что такие нарушения монотонности широко распространены и в обыденных рассуждениях. Создается впечатление, что люди часто не переходят, а “перепрыгивают” к заключениям. Например, увидев автомобиль, припаркованный на улице, человек обычно предполагает, что у этого автомобиля четыре колеса, даже если видит три. (Если вы чувствуете, что существование четвертого колеса действительно находится под сомнением, продумайте также вопрос о том, действительно ли являются настоящими три видимых колеса или они заменены изображениями на картонках, а сам автомобиль стоит на подставках.) И в самом деле, теория вероятностей позволяет уверенно прийти в заключению, что вероятность существования четвертого колеса весьма велика (хотя и не равна единице).

<sup>12</sup> Напомним, что условие монотонности требует, чтобы все высказывания, которые следуют из базы знаний KB, по-прежнему следовали бы из нее после добавления новых высказываний. Это означает, что если  $KB \models \alpha$ , то  $KB \wedge \beta \models \alpha$ .

це), но большинству людей мысль о том, что у автомобиля нет четвертого колеса, даже не приходит в голову, если им не представляются новые факты, позволяющие изменить свое мнение. Таким образом, создается впечатление, что вывод о наличии у наблюдавшегося автомобиля четырех колес достигается по умолчанию, если нет каких-либо причин поставить этот вывод под сомнение. Если же поступают новые факты (например, становится очевидно, что владелец катит колесо, а сам автомобиль поднят домкратом), то первоначальный вывод может быть отброшен. Принято говорить, что в рассуждениях такого рода проявляется **немонотонность**, поскольку множество убеждений не возрастает монотонно со временем, по мере поступления новых фактов. Для того чтобы можно было описывать подобное поведение, были разработаны **немонотонные логики**, в которых используются модифицированные определения понятий истинности и логического следствия. В данном разделе рассматриваются две такие логики, для изучения которых были проведены обширные исследования: логика косвенного описания и логика умолчания.

**Косвенное описание** (*circumscription*) может рассматриваться как более мощная и точная версия предположения о замкнутом мире. Его идея состоит в том, что должны быть заданы конкретные предикаты, в отношении которых предполагается, что они являются “настолько ложными, насколько это возможно”, т.е. ложными для всех объектов, за исключением тех, для которых эти предикаты заведомо истинны. Например, предположим, что необходимо ввести в базу знаний применяемое по умолчанию правило, что птицы летают. Для этого введем предикат, скажем *Abnormal<sub>1</sub>(x)*, и запишем следующее:

$$\text{Bird}(x) \wedge \neg\text{Abnormal}_1(x) \Rightarrow \text{Flies}(x)$$

Если будет указано, что для предиката *Abnormal<sub>1</sub>* должно применяться **косвенное описание**, то программа формирования рассуждений на основе косвенных описаний получает право предполагать, что  $\neg\text{Abnormal}_1(x)$ , если не известно, что *Abnormal<sub>1</sub>(x)* является истинным. Это позволяет выводить заключение *Flies(Tweety)* из предпосылки *Bird(Tweety)*, но такое заключение становится недействительным, если в базу знаний вводится утверждение *Abnormal<sub>1</sub>(Tweety)*.

Косвенное описание может рассматриваться как один из примеров логики **предпочтения моделей** (*model preference*). В подобных логиках высказывание следует из базы знаний (со статусом, заданным по умолчанию), если оно истинно во всех предпочтительных моделях базы знаний; в этом данное требование отличается от требования истинности во всех моделях в классической логике. С точки зрения косвенного описания одна модель является предпочтительной по отношению к другой, если в ней имеется меньшее количество аномальных объектов<sup>13</sup>. Рассмотрим, как эта идея может применяться в контексте множественного наследования в семантических сетях. Стандартный пример, демонстрирующий проблемы множественного наследования, называется “парадоксом Никсона”. Этот пример основан на том наблюдении, что Ричард Никсон был одновременно и квакером (поэтому по

<sup>13</sup> С точки зрения предположения о замкнутом мире одна модель является предпочтительной по отношению к другой, если в ней имеется меньше истинных атомов, т.е. предпочтительными моделями являются **минимальные** модели. Существует естественная связь между предположением о замкнутом мире и базами знаний с определенными выражениями, поскольку фиксированная точка, достигаемая в процессе прямого логического вывода в такой базе знаний, представляет собой уникальную минимальную модель (см. с. 315).

умолчанию пацифистом), и республиканцем (поэтому по умолчанию не пацифистом). Эту ситуацию можно описать следующим образом:

$$\begin{aligned} & \text{Republican}(\text{Nixon}) \wedge \text{Quaker}(\text{Nixon}) \\ & \text{Republican}(x) \wedge \neg \text{Abnormal}_2(x) \Rightarrow \neg \text{Pacifist}(x) \\ & \text{Quaker}(x) \wedge \neg \text{Abnormal}_3(x) \Rightarrow \text{Pacifist}(x) \end{aligned}$$

Если применяется косвенное описание для предикатов  $\text{Abnormal}_2$  и  $\text{Abnormal}_3$ , то возникают две предпочтительные модели: в одной из них истинны выражения  $\text{Abnormal}_2(\text{Nixon})$  и  $\text{Pacifist}(\text{Nixon})$ , а в другой —  $\text{Abnormal}_3(\text{Nixon})$  и  $\neg \text{Pacifist}(\text{Nixon})$ . Таким образом, программа формирования рассуждений на основе косвенных описаний остается в полном неведении в отношении того, является ли Никсон пацифистом. При желании можно дополнительно ввести утверждение, что религиозные убеждения имеют приоритет над политическими убеждениями; для этого можно воспользоваться формализмом, называемым **косвенным описанием с приоритетами** (prioritized circumscription), чтобы отдать предпочтение моделям, в которых минимизируется предикат  $\text{Abnormal}_3$ .

**Логика умолчания** (default logic) — это формальная система, в которой могут быть записаны **применяемые по умолчанию правила**, применяемые для вывода непротиворечивых немонотонных заключений. Заданное по умолчанию правило выглядит примерно таким образом:

$$\text{Bird}(x) : \text{Flies}(x) / \neg \text{Flies}(x)$$

Это правило означает, что если выражение  $\text{Bird}(x)$  является истинным, а выражение  $\text{Flies}(x)$  не противоречит базе знаний, то вывод  $\text{Flies}(x)$  может быть сделан по умолчанию. В общем случае заданное по умолчанию правило выглядит следующим образом:

$$P : J_1, \dots, J_n / C$$

где  $P$  называется предпосылкой,  $C$  — заключением, а  $J_i$  представляют собой обоснования; если можно доказать, что любое из них ложно, то нельзя вывести заключение. Любая переменная, которая появляется в  $J_i$  или  $C$ , должна также находиться и в  $P$ . Пример с парадоксом Никсона может быть представлен в логике умолчаний с помощью одного факта и двух заданных по умолчанию правил следующим образом:

$$\begin{aligned} & \text{Republican}(\text{Nixon}) \wedge \text{Quaker}(\text{Nixon}) \\ & \text{Republican}(x) : \neg \text{Pacifist}(x) / \neg \text{Pacifist}(x) \\ & \text{Quaker}(x) : \text{Pacifist}(x) / \text{Pacifist}(x) \end{aligned}$$

Для интерпретации того, что означают заданные по умолчанию правила, определим понятие **расширения** теории умолчаний как максимального множества следствий из этой теории. Таким образом, расширение  $S$  состоит из первоначально известных фактов и множества заключений, полученных на основе заданных по умолчанию правил, таких, что из  $S$  нельзя больше вывести дополнительные заключения, а обоснования всех сделанных по умолчанию заключений в  $S$  не противоречат  $S$ . Как и в случае предпочтительных моделей, в логике косвенного описания существуют два возможных расширения для парадокса Никсона: согласно одному из них он является пацифистом, а согласно другому — нет. Имеются также схемы с приоритетами, в которых определенные заданные по умолчанию правила могут получать преимущества над другими, что позволяет разрешать некоторые противоречия.

С 1980 года, когда впервые были предложены немонотонные логики, был достигнут большой прогресс в понимании их математических свойств. А начиная с последней половины 1990-х годов появились практические системы, основанные на логическом программировании, которые продемонстрировали перспективность их использования в качестве инструментальных средств представления знаний. Тем не менее остаются нерешенными некоторые вопросы. Например, если высказывание: “Автомобили имеют четыре колеса” является ложным, то что влечет за собой наличие такого высказывания в некоторой базе знаний? Каковым является наиболее приемлемое множество заданных по умолчанию правил? Если нет возможности принять отдельно для каждого правила решение о том, должно ли оно находиться в нашей базе знаний, то налицо серьезная проблема отсутствия модульности. Наконец, как могут использоваться для принятия решений определенные убеждения, имеющие заданный по умолчанию статус? По-видимому, эта проблема является наиболее сложной для систем формирования рассуждений по умолчанию. Принятие решений часто связано с поиском компромиссов, и поэтому необходимо сравнивать силу убеждений в отношении предполагаемых результатов различных действий. В тех случаях, когда решения одного и того же типа должны быть приняты повторно, появляется возможность интерпретировать заданные по умолчанию правила как высказывания с “пороговой вероятностью”. Например, заданное по умолчанию правило “Тормоза в моем автомобиле всегда в порядке” фактически означает “Если нет другой информации, то вероятность, что тормоза в моем автомобиле в порядке, достаточно велика, чтобы оптимальным решением для меня был выезд на дорогу без их проверки”. Если контекст принятия решения изменяется (например, если приходится спускать тяжело груженый самосвал вниз по крутой горной дороге), такое заданное по умолчанию правило внезапно становится неприемлемым, даже если нет новых фактов, говорящих о том, что тормоза неисправны. Подобные соображения привели некоторых исследователей к выводу, что необходимо продумать, как внедрить средства формирования рассуждений по умолчанию в теорию вероятностей.

## 10.8. СИСТЕМЫ ПОДДЕРЖКИ ИСТИННОСТИ

В предыдущем разделе было показано, что многие логические выводы, полученные с помощью той или иной системы представления знаний, могут иметь лишь некоторый заданный по умолчанию статус, а не быть абсолютно достоверными. Поэтому некоторые из таких полученных логическим путем фактов неизбежно оказываются ложными и должны быть отброшены на основании новой информации. Этот процесс называется **пересмотром убеждений** (belief revision)<sup>14</sup>. Предположим, что база знаний *KB* содержит высказывание *P* (возможно, заключение, сформированное по умолчанию с помощью алгоритма прямого логического вывода, или, возможно, просто неверное утверждение) и требуется выполнить операцию *Tell(KB,  $\neg P$ )*.

<sup>14</sup> Пересмотр убеждений часто противопоставляется **обновлению убеждений** (belief update), которое происходит, когда осуществляется пересмотр базы знаний для того, чтобы она отражала какое-то изменение в мире, а не добавление новой информации о неизменном мире. В обновлении убеждений объединяется пересмотр убеждений с рассуждениями о времени и изменениях; процесс обновлений связан также с процессом **фильтрации**, описанным в главе 15.

Но для предотвращения возникновения противоречия необходимо вначале выполнить операцию  $\text{Retract}(KB, P)$ . На первый взгляд в этом нет ничего сложного. Но если на основании  $P$  были выведены какие-то дополнительные высказывания и внесены в базу знаний, то возникают проблемы. Например, импликация  $P \Rightarrow Q$  могла использоваться для внесения в базу знаний высказывания  $Q$ . Очевидное “решение” (извлечение всех высказываний, которые следуют из  $P$ ) неприемлемо, поскольку подобные высказывания могут иметь другие обоснования, помимо  $P$ . Например, если в базе знаний имеются также высказывания  $R$  и  $R \Rightarrow Q$ , то  $Q$  в конечном итоге вообще не следует удалять.  $\bowtie$  **Системы поддержки истинности**, или TMS (Truth Maintenance System), предназначены именно для того, чтобы можно было проще справиться с подобными осложнениями.

Один из очень простых подходов к созданию системы поддержки истинности состоит в том, чтобы следить за порядком, в котором высказывания вводятся в базу знаний, путем присваивания им номеров от  $P_1$  до  $P_n$ . После того как формируется вызов  $\text{Retract}(KB, P_i)$ , система возвращается к состоянию, непосредственно предшествующему добавлению высказывания  $P_i$ , удаляя тем самым  $P_i$ , и любые результаты логического вывода, полученные на основании  $P_i$ . После этого могут быть снова добавлены высказывания от  $P_{i+1}$  до  $P_n$ . Такая организация работы является простой и гарантирует, что база знаний всегда будет оставаться непротиворечивой, но для извлечения  $P_i$  требуется извлечение и повторная вставка  $n-i$  высказываний, а также отмена и повторное выполнение всех логических выводов, вытекающих из этих высказываний. Для систем, в которые происходит добавление многих фактов (таких как крупные коммерческие базы данных), указанный подход является практически не применимым.

Более эффективный подход состоит в создании системы поддержки истинности на основе обоснований, или системы  $\bowtie$  JTMS (Justification-Based Truth Maintenance System). В системе JTMS к каждому высказыванию в базе знаний прилагается аннотация в виде  $\bowtie$  обоснования, состоящего из множества высказываний, на основании которых было выведено это высказывание. Например, если база знаний уже содержит высказывание  $P \Rightarrow Q$ , то операция  $\text{Tell}(P)$  вызовет добавление  $Q$  с обоснованием  $\{P, P \Rightarrow Q\}$ . Вообще говоря, высказывание может иметь любое количество обоснований. Обоснования используются для обеспечения эффективного извлечения. После выполнения вызова  $\text{Retract}(P)$  система JTMS удалит такие и только такие высказывания, для которых  $P$  является элементом каждого обоснования. Поэтому, если высказывание  $Q$  имеет единственное обоснование  $\{P, P \Rightarrow Q\}$ , оно будет удалено; если имеет также дополнительное обоснование  $\{P, P \vee R \Rightarrow Q\}$ , оно также будет удалено; но если, кроме этого, имеет обоснование  $\{R, P \vee R \Rightarrow Q\}$ , оно будет сохранено. Таким образом, время, требуемое для извлечения высказывания  $P$ , зависит только от количества высказываний, полученных на основании  $P$ , а не от количества других высказываний, добавленных после того, как  $P$  было введено в базу знаний.

В системе JTMS предполагается, что высказывания, которые уже когда-то рассматривались, по-видимому, будут рассматриваться снова, поэтому вместо полного удаления из базы знаний некоторого высказывания после того, как оно теряет все обоснования, это высказывание просто отмечается как находящееся *вне* базы знаний. Если же какое-то последующее утверждение восстанавливает одно из обосно-

ваний, то это высказывание снова отмечается как находящееся *внутри* базы. Благодаря этому система JTMS позволяет сохранить все цепочки логического вывода, которые в ней используются, и не нуждается в повторном выводе высказываний после того, как некоторое обоснование вновь становится действительным.

Кроме успешного исключения из базы знаний неправильной информации, системы поддержки истинности могут также использоваться для ускорения анализа многочисленных гипотетических ситуаций. Предположим, например, что Олимпийский комитет Румынии выбирает площадки для проведения соревнований по плаванию, легкой атлетике и конному спорту для олимпийских игр 2048 года, которые должны проводиться в Румынии. Например, допустим, что первой гипотезой является следующая: *Site(Swimming, Pitesti)*, *Site(Athletics, Bucharest)* и *Site(Equestrian, Arad)*. В таком случае необходимо провести большой объем рассуждений, чтобы определить логические следствия, а значит и целесообразность этого выбора. Если же вместо этого потребуется рассмотреть вариант *Site(Athletics, Sibiu)*, то такая система TMS позволит избавиться от необходимости начинать всю эту работу с нуля. Вместо этого достаточно будет просто извлечь гипотезу *Site(Athletics, Bucharest)* и ввести *Site(Athletics, Sibiu)*, после чего система TMS возьмет на себя весь необходимый пересмотр. А цепочки логического вывода, выработанные на основании выбора площадки в Бухаресте, могут повторно использоваться для Сибиу, при условии, что логические заключения остаются теми же самыми.

Особенно эффективное переключение контекста между гипотетическими мирами обеспечивает система поддержки истинности на основе предположения, или **ATMS** (Assumption-based Truth Maintenance System), которая предназначена именно для этой цели. В системе JTMS средства сопровождения обоснований позволяют быстро переходить от одного состояния к другому, выполнив лишь небольшой объем извлечений и вставок, но в них в любой момент времени представлено только одно состояние. А в системе ATMS представлены все состояния, которые когда-либо рассматривались одновременно. Это означает, что в системе JTMS каждое высказывание обозначается как находящееся вне или внутри базы знаний, тогда как в системе ATMS для каждого высказывания отслеживается, какие предположения могли бы вынудить это высказывание стать истинным. Иными словами, каждое высказывание имеет метку, состоящую из множества множеств предположений. Это высказывание становится истинным только в том случае, если истинными являются все предположения в одном из множеств предположений.

Системы поддержки истинности предоставляют также механизм выработки **объяснений**. Формально объяснением высказывания *P* является такое множество высказываний *E*, что из *E* следует *P*. Если уже известно, что высказывания *E* истинны, то *E* просто предоставляет достаточную базу для доказательства того, что *P* также является таковым. Но объяснения могут также включать **предположения** — высказывания, в отношении которых неизвестно, являются ли они истинными, но которые были бы достаточными, чтобы доказать истинность *P*, если бы они были истинными. Например, некто может не иметь достаточной информации, чтобы доказать, что двигатель его автомобиль не запустится, но обоснованное объяснение может включать предположение, что аккумулятор разряжен. Это позволяет объяснить наблюдаемое неправильное поведение двигателя, в сочетании со знаниями о

том, как работает двигатель автомобиля. В большинстве случаев предпочтительным является минимальное объяснение  $E$ ; под этим подразумевается, что не существует строгое подмножество  $E$ , которое также было бы объяснением. Система ATMS может сформировать объяснения для проблемы “двигатель автомобиля не запускается”, делая предположения (такие как “в карбюратор попал бензин” или “разряжен аккумулятор”) в любом желательном для пользователя порядке, даже если некоторые предположения противоречат друг другу. После этого достаточно посмотреть на метку, предусмотренную для высказывания “двигатель автомобиля не запускается”, чтобы ознакомиться с множествами предположений, которыми могло бы оправдываться это высказывание.

Точные алгоритмы, используемые для реализации систем поддержки истинности, являются довольно сложными, и в этой главе они не рассматриваются. Вычислительная сложность задачи поддержки истинности является, по меньшей мере, такой же, какая характерна для пропозиционального логического вывода, т.е. NP-трудной. Поэтому не следует рассчитывать на то, что подход на основе поддержки истинности окажется панацеей. Однако при их продуманном использовании системы TMS могут обеспечить существенное повышение способности логических систем действовать в условиях применения сложных гипотез и вариантов среды.

## 10.9. РЕЗЮМЕ

Эта глава оказалась наиболее подробной по сравнению со всеми предыдущими главами данной книги. Описывая детали того, какие способы используются для представления различных знаний, авторы надеялись дать читателю возможность понять, как создаются реальные базы знаний. Основные идеи, высказанные в этой главе, перечислены ниже.

- Для крупномасштабного представления знаний требуется онтология общего назначения, позволяющая организовать и связать воедино различные специализированные области знаний.
- Онтология общего назначения должна охватывать широкий круг знаний и быть способной, в принципе, представить любую проблемную область.
- В главе описана **верхняя онтология**, основанная на категориях и исчислении событий. Здесь рассматривались структурированные объекты, время и пространство, изменения, процессы, вещества и убеждения.
- Действия, события и время могут быть представлены либо с помощью ситуационного исчисления, либо с применением более выразительных средств представления, таких как исчисление событий и исчисление флюентных высказываний. Такие представления дают возможность агенту составлять планы с помощью логического вывода.
- Мыслительные состояния агентов могут быть представлены строками, которые описывают их убеждения.
- В главе представлен подробный анализ проблемной области совершения покупок в Internet, исследована общая онтология и показано, как знания проблемной области могут использоваться торговым агентом.

- Описаны системы представления общего назначения, такие как **семантические сети и описательные логики**, позволяющие организовывать иерархии категорий. С этими системами связана важная форма логического вывода, называемая **наследованием**, позволяющая определять логическим путем свойства объектов на основании данных об их принадлежности к категориям.
- **Предположение о замкнутом мире**, будучи реализованным в логических программах, предоставляет простой способ избежать необходимости задавать большие объемы отрицательной информации. Такую информацию проще всего интерпретировать как **заданную по умолчанию**, которая может быть переопределена с помощью дополнительной информации.
- В целом для предоставления возможности осуществлять формирование рассуждений по умолчанию предназначены **немонотонные логики**, такие как **логика косвенного описания** и **логика умолчания**. Применение **программирования множества ответов** позволяет ускорить немонотонный логический вывод, во многом аналогично тому, как использование алгоритма WalkSAT ускоряет пропозициональный логический вывод.
- **Системы поддержки истинности** позволяют эффективно осуществлять обновления и пересмотры баз знаний.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Многие исследователи вполне обосновано утверждают [186], что исследования в области формального представления знаний начались с классических работ индийских теоретиков, посвященных грамматике шастрнических текстов на санскрите, которые относятся к первому тысячелетию до н.э. На Западе к числу самых первых примеров таких исследований относится использование определений терминов в трудах древнегреческих математиков. Безусловно, как форма представления знаний может также рассматриваться разработка технической терминологии в любой области.

На первых порах дискуссии на тему представления в искусственном интеллекте в основном сосредоточивались на “представлении задач”, а не на “представлении знаний” (см., например, приведенное Амарелем [24] обсуждение задачи с миссионерами и каннибалами). В 1970-х годах основной акцент в искусственном интеллекте был сделан на разработке “экспертных систем” (называемых также “системами, основанными на знаниях”), которые были способны при наличии соответствующих знаний в проблемной области достичь или превзойти производительность людей-экспертов при решении узко определенных задач. Например, даже первая экспертная система, Dendral [458], [936], интерпретировала выходные данные массового спектрометра (прибора, применяемого для анализа структуры органических химических веществ) столь же точно, как и опытные химики. Хотя успех системы Dendral позволил сообществу исследователей по искусственному интеллекту убедиться в важности представления знаний, формальные средства представления, используемые в Dendral, были в высшей степени специализированными и относящимися только к данной проблемной области химии. Со временем у исследователей появилось стремление к стандартизации формальных средств представления знаний и онтологий, что позволило бы упростить процесс создания новых экспертных систем.

В ходе решения этой проблемы им пришлось проникнуть на территорию, которая до сих пор исследовалась только философами, изучающими проблемы науки и языка. Принятое в искусственном интеллекте требование, согласно которому предлагаемые теории должны “работать”, привело к более быстрому и глубокому прогрессу в решении проблем по сравнению с той ситуацией, когда эти проблемы рассматривались как принадлежащие исключительно к области философии (хотя такой подход вре́мья от времени приводил к “повторному изобретению колеса”).

Первые попытки создания всеобъемлющих таксономий, или классификаций, относятся к временам древности. Применения правильных схем классификации и категоризации настоятельно требовал Аристотель (384–322 до н.э.). В *Органон*, коллекцию трудов Аристотеля по логике, собранную учениками Аристотеля после его смерти, включен трактат под названием *Категории*, в котором Аристотель попытался создать то, что мы теперь называем *верхней онтологией*. Он также ввел понятия **родов и видов** для классификации нижнего уровня, хотя и не в их современном, сугубо биологическом смысле. Современная система биологической классификации, включая использование “биноминальной номенклатуры” (в формальном смысле слова — классификации по родам и видам), была предложена шведским биологом Карлом Линнеем (1707–1778). Проблемы, связанные с естественными разновидностями и неточными границами категорий, кроме многих других работ, рассматривались в [882], [1252], [1372] и [1608].

В последнее время продолжает расти интерес к крупномасштабным онтологиям. В проекте CYC [908], [910] была разработана верхняя онтология с 6000 понятий, состоящая из 60 000 фактов, а также лицензирована гораздо более крупная глобальная онтология. Организацией IEEE был создан подкомитет P1600.1, получивший название “Рабочей группы по стандарту верхней онтологии” (Standard Upper Ontology Working Group), а организация Open Mind Initiative привлекла к работе больше 7000 пользователей Internet, собрав свыше 400 000 фактов об обыденных понятиях. В среде Web появились такие стандарты, как RDF, XML и Semantic Web [110], хотя они еще не нашли широкого распространения. Много интересных статей в области общих и специализированных онтологий публикуется по материалам конференции *Formal Ontology in Information Systems* (FOIS).

Таксономия, используемая в этой главе, разработана авторами данной книги и частично основана на опыте их участия в проекте CYC, а также на работах Хванга, Шуберта [714] и Дэвиса [332]. Воодушевляющее описание общего проекта представления обыденных знаний приведено в книге Хейса *The Naive Physics Manifesto* [634], [637].

Проблемы представления времени, изменений, действий и событий широко исследовались не только в искусственном интеллекте, но и в философии и теоретических компьютерных науках. К числу наиболее развитых направлений относится **временная логика**, представляющая собой специализированную логику, в которой каждая модель описывает полную траекторию развития процесса во времени (обычно либо линейную, либо ветвящуюся), а не просто статическую реляционную структуру. Эта логика включает **модальные операторы**, применяемые к формулам;  $\Box p$  означает “ $p$  будет истинным во все времена в будущем”, а  $\Diamond p$  означает “ $p$  станет истинным в некоторый момент в будущем”. Исследования в области временной логики были впервые начаты Аристотелем, а также представителями мегарской и стоической школ в древней Греции. В современной истории развития науки применение формального исчисления для проведения рассужде-

ний о времени было впервые предложено Финдлеем [468], но считается, что наибольшее влияние оказала работа Артура Прайора [1239]. К учебникам по временной логике относятся [1283] и [1529].

Специалисты в области теоретических компьютерных наук уже давно стремились формально описать свойства программ, рассматриваемых как последовательности вычислительных действий. Бурстолл [210] выдвинул идею использования модальных операторов для формирования рассуждений о компьютерных программах. Вскоре после этого Боган Пратт [1233] разработал **динамическую логику**, в которой модальные операторы обозначают результаты выполнения программ или другие действия (см. также [620]). Например, в динамической логике, если  $\alpha$  — имя программы, то “[ $\alpha$ ]  $p$ ” означает “ $p$  должно быть истинным во всех состояниях мира, ставших результатом выполнения программы  $\alpha$  в текущем состоянии мира”, а “[ $\langle \alpha \rangle p$ ” означает “ $p$  должно быть истинным по меньшей мере в одном из состояний мира, ставшим результатом выполнения программы  $\alpha$  в текущем состоянии мира”. Динамическая логика была применена для фактического анализа программ Фишером и Ладнером [472]. В [1218] предложена идея использования классической временной логики для формирования рассуждений о программах.

Временная логика предусматривает введение времени непосредственно в теорию моделей языка, а методы представления времени, применяемые в искусственном интеллекте, как правило, предусматривали явное включение аксиом об интервалах времени и событиях в базу знаний, поэтому время не получало особого статуса в этой логике. Кроме того, данный подход в некоторых случаях обеспечивает большую ясность и гибкость. К тому же знания о времени, выраженные в логике первого порядка, могут быть более легко интегрированы с другими знаниями, накопленными в рассматриваемой проблемной области.

Первой попыткой представления времени и действий в искусственном интеллекте стало ситуационное исчисление, предложенное Джоном Маккарти [1010]. Первой системой искусственного интеллекта, в которой широко применялись рассуждения общего назначения о действиях в логике первого порядка, была система QA3 [592]. Ковалевский [851] развил идею овеществления высказываний в рамках ситуационного исчисления.

**Проблема окружения** была впервые описана в [1013]. Многие исследователи считали эту проблему неразрешимой в логике первого порядка, и она стала стимулом для проведения большого объема исследований в области немонотонных логик. Многие философы, начиная от Дрейфуса [414] и заканчивая Крокеттом [310], указывали, что проблема окружения является одним из симптомов неизбежного краха всего направления работ по созданию искусственного интеллекта. Частичное решение проблемы представительного окружения с использованием аксиом состояния-преемника было предложено Рэем Рейтером [1277]; истоки решения проблемы выводимого окружения прослеживаются в [671], в которой были предложены методы, получившие название *исчисления флюентных высказываний* [1504]. Описание, приведенное в настоящей главе, частично основано на результатах анализа, изложенных в [931] и [1504]. Книги Шенахана [1391] и Рейтера [1279] содержат полное, современное изложение вопросов формирования рассуждений о действиях в ситуационном исчислении.

Частичное разрешение проблемы окружения снова пробудило стремление к использованию декларативного подхода для формирования рассуждений о действиях,

что привело к созданию многих систем планирования специального назначения, начиная с первой половины 1970-х годов (см. главу 11). В исследованиях, проводимых в рамках ~~когнитивной робототехники~~, был достигнут большой прогресс в области разработки средств логического представления действий и времени. В языке Golog используется полная выразительная мощь логического программирования для описания действий и планов [917], к тому же этот язык был дополнен, для того чтобы в нем можно было представлять параллельные действия [550], стохастические варианты среды [160] и результаты восприятия [1278].

Для создания средств представления непрерывного времени было предложено использовать исчисление событий, разработанное Ковальским и Серготом [854], а в дальнейшем появилось еще несколько вариантов исчисления событий [1341]. В [1392] представлен хороший краткий обзор. Джеймс Аллен предложил использовать для этой же цели временные интервалы [15], [16], указывая, что интервалы являются гораздо более естественным средством формирования рассуждений о продолжительных и одновременных событиях, чем ситуации. В [877], [878] впервые предложены “вогнутые” временные интервалы (интервалы с перерывами; по сути объединения обычных “выпуклых” временных интервалов) и для представления времени применены математические методы абстрактной алгебры. Аллен [17] систематически исследовал широкий спектр методов, которые могут применяться для представления времени. Шохем [1403] описал процедуру овеществления событий и предложил использовать для этой цели разработанную им новейшую схему. Между онтологией на основе событий, приведенной в данной главе, и анализом событий, выполненным философом Дональдом Давидсоном [328], имеются весьма важные аналогии. К тому же типу относятся хронологии (history), предложенные в работе Патрика Хейса [636]; аналогичной разновидностью представления во многом является также онтология вневременных событий (liquid event).

Проблемы онтологического статуса веществ имеют длинную историю. Платон считал, что вещества — это абстрактные сущности, полностью отличающиеся от физических объектов; с его точки зрения следовало бы сказать, что кусок масла сделан из масла,  $MadeOf(Butter_3, Butter)$ , а не что кусок масла является элементом множества масла,  $Butter_3 \in Butter$ . Такая идея ведет к созданию иерархии веществ, в которой, например, несоленое масло *UnsaltedButter* является более конкретно определенным веществом, чем само масло *Butter*. Научная позиция, принятая в этой главе, согласно которой вещества представляют собой категорию объектов, была обоснована Ричардом Монтегио [1072]. Кроме того, эта позиция была принята и в проекте CYC. В [293] на эту позицию предпринята серьезная, но не настолько уж неотразимая атака. Альтернативный подход, упомянутый в данной главе, согласно которому масло представляет собой единственный объект, состоящий из всех маслоподобных объектов во вселенной, был первоначально предложенпольским логиком Лесьневским [913]. В разработанной им ~~мереологии~~ (это название происходит от греческого слова, обозначающего “часть”) используется отношение “часть—целое” в качестве замены математической теории множеств, в целях устранения таких абстрактных сущностей, как множества. Более удобное для чтения изложение этих идей приведено в [911], а в книге Гудмана *The Structure of Appearance* [579] эти идеи применяются для решения различных проблем в области представления знаний. Хотя в некоторых аспектах мереологический подход является весьма громоздким (например, в нем требуется отдельный механизм наследования, осно-

ванный на отношениях “часть—целое”), он получил поддержку Квайна [1253]. Гарри Бант [207] провел широкий анализ перспектив использования этого подхода в области представления знаний.

Мыслимые объекты и мыслительные состояния были предметом интенсивных исследований в области философии и искусственного интеллекта. **Модальная логика** представляет собой классический метод формирования рассуждений о знаниях, применяемый в философии. В модальной логике логика первого порядка дополняется модальными операторами, такими как *B* (от *believes* — убежден) и *K* (от *knows* — знает), которые принимают в качестве своих параметров высказывания, а не термы. В теории доказательств для модальной логики жестко регламентируются подстановки в пределах модальных контекстов, что позволяет обеспечить ссылочную непрозрачность. Модальная логика знаний была предложена Яакко Хинтиккой [654]. Саул Кripке [859] определил семантику модальной логики знаний в терминах **возможных миров**. Грубо говоря, мир для агента является возможным, если не противоречит всему, что знает агент. На основании такого подхода могут быть разработаны правила логического вывода с использованием оператора *K*. Роберт К. Мур связал модальную логику знаний с таким стилем формирования рассуждений о знаниях, который позволяет непосредственно ссылаться на возможные миры в логике первого порядка [1080], [1081]. Модальная логика может на первый взгляд показаться устрашающе запутанной областью науки, но она нашла очень важные применения в области формирования рассуждений об информации в распределенных компьютерных системах. В книге Фейгина и др. *Reasoning about Knowledge* [449] приведено исчерпывающее введение в этот модальный подход. Приложения теории знаний в искусственном интеллекте, экономике и распределенных системах обсуждаются на проводимой один раз в два года конференции *Theoretical Aspects of Reasoning About Knowledge* (TARK).

Синтаксическая теория мыслимых объектов была впервые глубоко исследована Капланом и Монтею [770], которые показали, что эта теория может приводить к парадоксам, если при использовании ее средств не соблюдается чрезвычайная осторожность. Поскольку такая теория позволяет создавать естественные модели физических конфигураций компьютера или мозга в терминах убеждений, она в последние годы получила широкое распространение в области искусственного интеллекта. В [604] и [833] эта теория использовалась для описания машин логического вывода с ограниченной мощью, а Моргенштерн [1086] показал, как можно ее использовать для описания предусловий знаний в планировании. Методы планирования таких действий, как наблюдение, описанные в главе 12, основаны на этой синтаксической теории. Эрни Дэвис [332] дал превосходное сравнение синтаксической и модальной теорий знания.

Греческий философ Порфирий (ок. 234–305 н.э) в своих комментариях к трактату Аристотеля *Категории* продемонстрировал то, что может рассматриваться как первая семантическая сеть. Чарльз С. Пирс [1199] разработал экзистенциальные графы, которые могут рассматриваться как первое формальное определение семантической сети с использованием современной логики. Инициатором исследований по семантическим сетям, проводимых в рамках искусственного интеллекта, был Росс Квиллиан [1251], основным стимулом для которого был интерес к человеческой памяти и языковой обработке. В важной статье Марвина Минского [1053] представлена одна из версий семантических сетей, основанная на использовании

так называемых **фреймов**; фреймы служили для представления объектов или категорий и характеризовались наличием атрибутов и отношений с другими объектами или категориями. Хотя эта статья послужила важной причиной пробуждения интереса к области представления знаний как таковой, она подверглась критике за то, что в ней просто под другим углом были изложены идеи статей [132], [320], разработанные ранее в объектно-ориентированном программировании, такие как наследование и использование заданных по умолчанию значений. Однако еще не совсем ясно, в какой степени на указанные статьи по объектно-ориентированному программированию, в свою очередь, повлияли еще более ранние работы в области искусственного интеллекта, посвященные семантическим сетям.

Проблемы семантики приобрели особую остроту применительно к семантическим сетям, разработанным Квиллианом (и теми, кто стал последователем предложенного им подхода), в связи с тем, что в них использовались вездесущие и весьма неопределенные “связи IS-A”, а также применительно к другим ранним формальными системам представления знаний, таким как Merlin [1079], с ее загадочными операциями “flat” и “cover”. Знаменитая статья Вудса “*What's In a Link?*” [1613] привлекла внимание исследователей в области искусственного интеллекта к тому, что в формальных системах представления знаний должна быть точно определена семантика. Брачман [167] провел исследования по этой проблеме и предложил некоторые решения. Патрик Хейс в своей книге *The Logic of Frames* [635] провел еще более глубокие исследования и сформулировал утверждение, что “так называемые «фреймы» по большей части представляют собой просто новые синтаксические обозначения для фрагментов логики первого порядка”. Дрю Макдермотт в своей работе *Tarskian Semantics, or, No Notation without Denotation!* [1022] доказывал, что модельно-теоретический подход к семантике, используемый в логике первого порядка, должен быть распространен на все формальные системы представления знаний. Однако эта идея является внутренне противоречивой; замечательно то, что сам Макдермотт пересмотрел свою позицию в работе *A Critique of Pure Reason* [1023]. Проявлением нового подхода стала Netl [451], сложная система семантической сети, в которой связи IS-A (называемые связями “виртуальной копии”, или VC — virtual copy) были основаны в большей степени на понятии характеристик “наследования” систем фреймов или объектно-ориентированных языков программирования, чем на отношении подмножества, и были определены гораздо более точно по сравнению со связями, применявшимися Квиллианом в эпоху, предшествовавшую появлению работ Вудса. Система Netl привлекла особое внимание, поскольку предназначалась для реализации в параллельном аппаратном обеспечении для преодоления сложностей выборки информации из больших семантических сетей. Дэвид Турецкий [1512] подверг понятие наследования строгому математическому анализу. В [1381] обсуждаются сложности наследования с исключениями и показано, что в большинстве формулировок задача представления наследования является NP-полной.

Разработка описательных логик представляет собой один из наиболее современных этапов в длинной цепи исследований, нацеленных на поиск полезных подмножеств логики первого порядка, для которой задача логического вывода является осуществимой с помощью вычислительных средств. Гектор Левеск и Рон Брачман [916] показали, что неразрешимость логического вывода в основном возникает из-за использования определенных логических конструкций (особенно отличаются этим некоторые варианты использования дизъюнкции и отрицания). На основе системы

KL-One [1362] был разработан целый ряд систем, в проектах которых применены результаты теоретического анализа сложности; к числу наиболее известных из них относятся Krypton [168] и Classic [155]. Полученные при этом результаты продемонстрировали заметное повышение скорости логического вывода и позволили гораздо лучше понять зависимости между сложностью и выразительностью в системах формирования рассуждений. Общий итог современного состояния дел в этой области подведен в [215]. Протестуя против этой тенденции к упрощению, Дойл и Петил [411] доказывали, что ограничение выразительности языка либо исключает возможность решения некоторых проблем, либо побуждает пользователя обходить ограничения языка с помощью нелогических средств.

Все три основные формальные системы, предназначенные для использования в немонотонном логическом выводе, — косвенное описание [1012], логика умолчания [1276] и модальная немонотонная логика [1025] — были предложены в одном специальном выпуске *AI Journal*. Программирование множества ответов может рассматриваться как расширение понятия отрицания как недостижения цели или как уточнение понятия косвенного описания; основополагающая теория семантики стабильных моделей была предложена в [533], а ведущими системами программирования множества ответов являются DLV [432] и Smodels [1138]. Пример с дисковым накопителем взят из руководства пользователя Smodels [1484]. В [929] обсуждается использование программирования множества ответов для планирования. В [184] приведен хороший краткий обзор различных подходов к использованию немонотонной логики. В [262] рассматривается подход к логическому программированию, основанный на отрицании как недостижении цели, и анализируется дополнение Кларка. Ван Эмден и Ковальский [1530] показали, что каждая программа Prolog без отрицаний имеет уникальную минимальную модель. Последние годы характеризуются повторным пробуждением интереса к приложениям немонотонных логик в крупномасштабных системах представления знаний. По-видимому, первой системой, достигшей коммерческого успеха, является система BenInq для обработки данных опросов, касающихся страховых льгот; эта система представляет собой приложение системы немонотонного наследования [1087]. В [929] обсуждается применение программирования множества ответов в планировании. Целый ряд систем формирования немонотонных рассуждений, основанных на логическом программировании, описан в трудах конференции *Logic Programming and Nonmonotonic Reasoning* (LPNMR).

Исследования в области систем поддержки истинности начались с создания систем TMS [409] и RUP [1003], которые по существу представляли собой системы JTMS. Подход на основе систем ATMS был описан в ряде статей Йохана де Клеера [345–347]. В работе *Building Problem Solvers* [479] подробно объяснено, как могут применяться системы TMS в приложениях искусственного интеллекта. В [1117] показано, что использование эффективной системы TMS обеспечило возможность планирования операций космического корабля агентства NASA в реальном времени.

По очевидным причинам в данной главе подробно не рассматривается каждая область представления знаний. Ниже описаны три основные темы, которые в ней не представлены.

-  **Качественная физика.** Это подобласть представления знаний, которая главным образом относится к формированию логической, нечисловой теории фи-

зических объектов и процессов. Этот термин был предложен Йоханом де Клеером [344], хотя вполне можно считать, что это направление исследований началось с создания Фалманом [450] программы Build — развитого планировщика для построения сложных башен из блоков. В процессе проектирования этой программы Фалман открыл, что основная часть усилий (по его оценке, 80%) уходит на моделирование физических свойств мира блоков в целях вычисления устойчивости различных субфрагментов структур блоков, а не на само планирование как таковое. Он сформулировал набросок гипотетического процесса, подобного проведению рассуждений в рамках наивной физики, для объяснения причин того, почему дети младшего возраста способны решать задачи, подобные рассматриваемым в программе Build, без обращения к быстroredействующей арифметике с плавающей точкой, которая используется в физическом моделировании программой Build. Хейс [636] применил “истории” (четырехмерные фрагменты пространства-времени, аналогичные событиям Дэвидсона) для построения довольно сложной наивной физической теории жидкостей. Хейсу впервые удалось доказать в рамках искусственного интеллекта, что ванна с заткнутым сливом в конечном итоге переполнится, если вода из крана будет продолжать бежать, и что человек, упавший в озеро, весь вымокнет. В [349] и [478] описаны попытки создать нечто подобное общей теории физического мира на основе качественных абстракций физических уравнений. За последние годы качественная физика была разработана до такой степени, что с ее помощью стало возможным анализировать сложные физические системы, разнообразие которых весьма впечатляет [1340], [1631]. Качественные методы использовались для разработки новейших проектов часов, дворников для ветрового стекла и шестиногих шагающих машин [1470], [1472]. Хорошим введением в эту область является сборник статей *Readings in Qualitative Reasoning about Physical Systems* [1570].

- **❖ Пространственные рассуждения.** Рассуждения, необходимые для навигации в мире вампуса и в мире совершения покупок, являются тривиальными по сравнению с теми, что требуются в богатой пространственной структуре реального мира. Результаты самых первых серьезных попыток описать обыденные рассуждения о пространстве приведены в работах Эрнеста Дэвиса [331], [332]. В исчислении региональных связей Кона и др. [280] поддерживается определенная форма качественных пространственных рассуждений; эти исследования привели к созданию географических информационных систем нового типа. Как и с помощью качественной физики, агент, так сказать, многое может сделать без обращения к полному метрическому представлению. А если такое представление становится необходимым, то можно воспользоваться методами, разработанными в робототехнике (глава 25).
- **❖ Психологические рассуждения.** Психологические рассуждения касаются проблемы разработки для искусственных агентов работоспособных психологических моделей, которые могли бы использоваться агентами для формирования рассуждений о себе и о других агентах. Такие рассуждения часто основаны на так называемой народной психологии (*folk psychology*) — теории, претендующей на понимание того, какие принципы используются людьми в рассуждениях о себе и других людях. Когда исследователи в области искусств-

венного интеллекта снабжают своих искусственных агентов психологическими теориями для формирования рассуждений о других агентах, эти теории часто основаны на описании исследователями проектов самих этих логических агентов. Поэтому в настоящее время наиболее продуктивны такие исследования в области психологических рассуждений в контексте понимания естественного языка, где наибольшую важность имеет предугадывание намерений говорящего.

Наиболее современными источниками сведений о работах в этих областях являются труды международной конференции *Principles of Knowledge Representation and Reasoning*. В работах *Readings in Knowledge Representation* [169] и *Formal Theories of the Commonsense World* [665] приведены превосходные антологии по представлению знаний; в первой в большей степени обсуждаются статьи, имевшие историческое значение в языках и формальных системах представления, а последняя посвящена накоплению самих знаний. В [332], [1448] и [1458] приведены учебные введения в проблематику представления знаний.

## УПРАЖНЕНИЯ

- 10.1.** Запишите высказывание, позволяющее определить результаты действия *Shoot* в мире вампуса. Опишите эти результаты применительно к вампусу, но не забудьте, что после выстрела у агента больше нет стрелы.
- 10.2.** С помощью ситуационного исчисления запишите аксиому, чтобы связать время 0 с ситуацией  $S_0$ , и еще одну аксиому, чтобы связать время  $t$  с любой ситуацией, которая развилась из  $S_0$  в результате последовательности из  $t$  действий.
- 10.3.** В данном упражнении рассматривается проблема планирования маршрута для робота, чтобы он мог добраться из одного города в другой. Основным действием, предпринимаемым роботом, является действие  $Go(x, y)$ , которое позволяет ему добраться из города  $x$  в город  $y$ , если между этими городами есть прямой путь. Предикат  $DirectRoute(x, y)$  принимает истинное значение тогда и только тогда, когда есть прямой путь из  $x$  в  $y$ ; вы можете принять предположение, что все соответствующие факты уже находятся в базе знаний (см. карту на с. 114). Робот начинает движение с Арада и должен достичь Бухареста.
  - a)** Запишите подходящее логическое описание начальной ситуации для робота.
  - б)** Запишите подходящий логический запрос, решения которого могут служить описаниями возможных путей к цели.
  - в)** Запишите высказывание, описывающее действие  $Go$ .
  - г)** Теперь предположим, что, следя по прямому пути между двумя городами, агент тратит топливо, количество которого равно расстоянию между этими городами. Робот начинает движение с полной загрузкой топлива. Дополните разработанное вами представление с учетом этих соображений. Предложенное вами описание действий должно быть таковым, чтобы сформулированный вами ранее запрос все еще приводил к получению осуществимых планов.

- д) Опишите начальную ситуацию и запишите новое правило (или правила), определяющее действие *Go*.
- е) Теперь допустим, что в некоторых узлах маршрута имеются также бензозаправочные станции, в которых робот может пополнить свой топливный бак. Дополните разработанное вами представление и запишите все правила, необходимые для описания бензозаправочных станций, включая действие *Fillup* (Заправка).

**10.4.** Исследуйте способы дополнения исчисления событий, позволяющие учитывать в нем одновременные события. Возможно ли избежать комбинаторного взрыва количества аксиом?

**10.5.** Представьте приведенные ниже семь высказываний, используя и дополняя средства представления, описанные в этой главе.

- а) Вода находится в жидкому состоянии при температуре от 0 до 100 градусов Цельсия.
- б) Вода кипит при температуре 100 градусов Цельсия.
- в) Вода в бутылке для воды, принадлежащей Джону, замерзла.
- г) Минеральная вода “Перрье” — это тоже вода.
- д) Джон держит в своей бутылке для воды минеральную воду “Перрье”.
- е) Все жидкости имеют определенную температуру замерзания.
- ж) Литр воды весит больше, чем литр алкоголя.

Теперь повторите это упражнение с использованием представления, основанного на мереологическом подходе, в котором, например, *Water* — это объект, содержащий в качестве частей всю воду в мире.

**10.6.** Запишите определения для следующих предикатов:

- а) *ExhaustivePartDecomposition*
- б) *PartPartition*
- в) *PartwiseDisjoint*

Эти определения должны быть аналогичными определениям для предикатов *ExhaustiveDecomposition*, *Partition* и *Disjoint*. Является ли, по вашему мнению, общезначимым выражение *PartPartition(s, BunchOf(s))*? Если да, докажите его общезначимость; в противном случае приведите контрпример и определите достаточные условия, при которых это выражение становится истинным.

**10.7.** Запишите множество высказываний, которое позволяет рассчитать цену отдельного помидора (или другого объекта), если указана цена за килограмм. Дополните эту теорию, чтобы иметь возможность рассчитать цену пакета помидоров.

**10.8.** Одна из альтернативных схем представления мер предусматривает применение функции единиц к абстрактному объекту длины. В такой схеме следует писать *Inches(Length(L<sub>1</sub>))=1.5*. Как выглядит эта схема по сравнению с той, что описана в данной главе? Заслуживают внимания такие вопросы, как аксиомы преобразования, имена для абстрактных количеств (такие как “50 долларов”) и сравнения абстрактных мер в различных единицах (например, “50 дюймов больше 50 сантиметров”).

- 10.9.** Разработайте средство представления для курсов обмена валют, которое допускает ежесуточные колебания курсов.
- 10.10.** Это упражнение касается связей между категориями событий и интервалами времени, в которые они происходят.
- Определите предикат  $T(c, i)$  в терминах отношений *During* и  $\in$ .
  - Дайте точное объяснение причин, по которым не требуются две различные системы обозначений для описания конъюнктивных категорий событий.
  - Дайте формальное определение для предикатов  $T(\text{OneOf}(p, q), i)$  и  $T(\text{Either}(p, q), i)$ .
  - Объясните, почему целесообразно иметь две формы отрицания событий, аналогичные двум формам дизъюнкции событий. Назовите их *Not* и *Never* и дайте им формальные определения.
- 10.11.** Определите предикат *Fixed*, где  $\text{Fixed}(\text{Location}(x))$  означает, что местонахождение объекта  $x$  постоянно во времени.
- 10.12.** Определите предикаты *Before*, *After*, *During* и *Overlap* с использованием предиката *Meet* и функции *Start* и *End*, но не функции *Time* или предиката  $<$ .
- 10.13.** В разделе 10.5 предикаты *Link* и *LinkText* использовались для описания связей между Web-страницами. Запишите определения для *Link* и *LinkText*, используя, кроме всего прочего, предикаты *InTag* и *GetPage*.
- 10.14.** Одна из составляющих процесса совершения покупок, которая не рассматривалась в этой главе, касается проверки совместимости товаров. Например, если клиент заказывает компьютер, будет ли этот компьютер совместимым с имеющимися у него периферийными устройствами? Если же оформляется заказ на цифровую камеру, то будут ли к ней прилагаться подходящие плата памяти и аккумулятор? Разработайте базу знаний, позволяющую принимать решение о том, является ли множество товаров совместимым, которая может использоваться для предоставления консультаций, касающихся замены или приобретения дополнительных товаров, если имеющиеся товары являются несовместимыми. Убедитесь в том, что эта база знаний может применяться по меньшей мере с одной номенклатурой товаров и что ее несложно дополнить для использования с другими номенклатурами.
- 10.15.** Введите правила, позволяющие расширить определение предиката *Name(s, c)* так, чтобы строка, подобная “laptop computer”, согласовывалась с соответствующей категорией имен, применяемых в самых различных магазинах. Попытайтесь добиться того, чтобы предложенное вами определение стало наиболее общим. Проверьте его, изучив десять оперативных магазинов и ознакомившись с тем, какие имена в них присвоены трем различным категориям. Например, применительно к категории портативных компьютеров авторы обнаружили имена “Notebooks”, “Laptops”, “Notebook Computers”, “Notebook”, “Laptops and Notebooks” и “Notebook PCs”. Некоторые из них могут быть описаны с помощью явно заданных фактов *Name*, а для других могут применяться правила образования множественного числа, оформления перечислений и т.д.

**10.16.** Полное решение проблемы неточных соответствий описаниям, сформулированным покупателем при совершении покупок, является очень сложным и требует применения полного спектра средств обработки естественного языка и методов выборки информации (см. главы 22 и 23). Одним из небольших шагов в этом направлении является предоставление пользователю возможности задавать минимальные и максимальные значения для различных атрибутов. Допустим, что мы требуем от покупателя, чтобы он использовал следующую грамматику для формулировки описаний товаров:

```

Description → Category [Connector Modifier]*
Connector → "with" | "and" | "," | "с" | "и"
Modifier → Attribute | Attribute Op Value
Op → "=" | ">" | "<"

```

Здесь *Category* обозначает категорию товара, *Attribute* — это некоторая характеристика, такая как “частота процессора” или “цена”, а *Value* — желаемое значение этой характеристики. Поэтому запрос “компьютер с частотой процессора по меньшей мере 2,5 ГГц по цене меньше \$1000” должен быть выражен так: “компьютер с частотой процессора  $> 2,5$  ГГц и ценой  $< \$1000$ ”. Реализуйте торгового агента, который принимает описания на этом языке.

**10.17.** В нашем описании процесса осуществления покупок в Internet упущен наиболее важный этап — фактическая покупка товара. Предоставьте формальное логическое описание этапа покупки с использованием исчисления событий. Иначе говоря, определите последовательность событий, которая происходит, когда покупатель приобретает товар по кредитной карточке, а затем в конечном итоге получает выставленный ему счет и ему вручается товар.

**10.18.** Опишите событие, в котором один объект обменивается на какой-то другой. Опишите процесс покупки как разновидность обмена, в которой одним из объектов, участвующих в обмене, является некоторая денежная сумма.

**10.19.** В двух предыдущих упражнениях подразумевается использование довольно примитивного понятия владения. Например, покупатель приступает к приобретению товара, овладев денежными купюрами. Такое описание процесса покупки становится неадекватным, если, например, деньги покупателя хранятся в банке, поскольку у него на руках больше нет какой-то конкретной совокупности денежных купюр, которыми он может немедленно распорядиться. Ситуация становится еще более сложной, если учесть возможность займа, сдачи в аренду, взятия в аренду и передачи в залог. Исследуйте различные обыденные и юридические понятия владения и предложите схему, с помощью которой их можно представить формально.

**10.20.** Вы должны разработать систему предоставления консультаций студентам отделения компьютерных наук в части того, какие курсы они должны пройти в течение некоторого продолжительного периода, чтобы выполнить условия учебной программы (исходите из тех требований, которые предъявляются в вашем учебном заведении). Прежде всего подготовьте словарь, позволяющий представить всю информацию, а затем представьте эту информацию; после этого примените подходящий запрос к системе, которая должна возвра-

тить в качестве решения приемлемую программу обучения. Вы должны предусмотреть в определенной степени необходимость учитывать подготовку отдельных студентов в том смысле, что система должна спрашивать, какие основные или эквивалентные курсы уже прошел студент, и не формировать программы, в которых повторяются эти курсы.

Предложите способы усовершенствования этой системы, например, для того, чтобы в ней учитывались сведения о предпочтениях студента, рабочей нагрузке, о том, какие преподаватели ему нравятся или не нравятся, и т.д. Рассматривая каждую разновидность знаний, объясните, как она может быть представлена логически. Способна ли ваша система легко учитывать эту информацию и находить наилучшую программу обучения для студента?

- 10.21.** На рис. 10.1 показаны верхние уровни иерархии для всех понятий. Дополните ее, чтобы включить максимально возможное количество реальных категорий. Удобным способом выполнения этого требования является описание всего, с чем приходится сталкиваться в повседневной жизни. К этому относятся объекты и события. Начните с утреннего пробуждения и последовательно учитывайте все, что вы видите, чего касаетесь, что делаете и о чем думаете. Например, случайно выбранный день может свестись к цепочке: музыка, новости, молоко, ходьба, поездка, заправка топливом, вестибюль университета, ковер, разговор, профессор Фейтмен, щипленок под острым соусом, обожженный язык, 7 долларов, солнце, дневная газета и т.д.

Вы должны подготовить единственную схему иерархии (на большом листе бумаги), а также список объектов и категорий с отношениями, которым соответствуют элементы каждой категории. Каждый объект должен находиться в некоторой категории, а каждая категория должна относиться к этой иерархии.

- 10.22.** (*Адаптировано на основании примера Дуга Лената (Doug Lenat.)*) Ваша задача состоит в том, чтобы представить в логической форме достаточный объем знаний, позволяющий ответить на ряд вопросов о следующем простом высказывании:

Вчера Джон отправился в супермаркет North Berkeley Safeway и купил два фунта помидоров и фунт говяжьего фарша.

Начните с попытки представить содержимое этого высказывания в виде ряда утверждений. Вы должны записать высказывания, которые имеют простую логическую структуру (например, утверждения, согласно которым объекты имеют определенные свойства, связанные определенными отношениями, и все объекты, удовлетворяющие некоторому свойству, удовлетворяют также какому-то другому свойству). Для начала может помочь получение ответов на приведенные ниже вопросы.

- Какие классы, объекты и отношения вам потребуются? Есть ли у них родительские объекты, сестринские объекты и т.д.? (Кроме всего прочего, вам может потребоваться учитывать упорядочение событий и временное упорядочение.)
- В каком месте эти объекты должны войти в более общую иерархию?
- Каковы ограничения и взаимосвязи между ними?
- Насколько подробно следует определять каждое из различных понятий?

Созданная вами база знаний должна предоставлять возможность найти ответ на список вопросов, который будет вскоре приведен. Некоторые вопросы касаются материала, явно сформулированного в этом кратком рассказе, но для большинства из них требуется, чтобы тот, кто на них отвечает, обладал другими фоновыми знаниями — умел читать между строк. Вам придется вспомнить о том, какого рода товары бывают в супермаркете, что связано с покупкой выбранных товаров, для чего могут использоваться покупки, и т.д. Попытайтесь сделать применяемое вами представление настолько общим, насколько это возможно. Приведем тривиальный пример: не следует вводить в базу знаний высказывание: “Люди покупают продукты в супермаркете Safeway”, поскольку это не позволяет учесть существование тех, кто покупает продукты в другом супермаркете. Нет смысла вводить информацию о том, что “Джо готовит спагетти с помидорами и говяжьим фаршем”, поскольку она вообще не позволит сделать какие-либо полезные выводы. Кроме того, не превращайте вопросы в ответы; например, в вопросе 10.22, *в* сказано: “Купил ли Джон какие-либо мясные продукты?”, а не “Купил ли Джон фунт говяжьего фарша?”

Наметьте цепочки рассуждений, которые позволили бы ответить на эти вопросы. В процессе этого вам несомненно потребуется создать дополнительные понятия, ввести дополнительные утверждения и т.д. Если это возможно, воспользуйтесь какой-то системой формирования логических рассуждений, чтобы продемонстрировать достаточность вашей базы знаний. Многие из составленных вами высказываний в реальности могут оказаться лишь приближенно правильными, но не следуют об этом слишком беспокоиться; идея состоит в том, чтобы извлечь обыденные знания (на уровне здравого смысла), которые, в принципе, позволяют вам ответить на эти вопросы. Задача составления действительно полного ответа на многие из этих вопросов является чрезвычайно трудной и, по-видимому, выходит за рамки современных средств представления знаний. Но вы должны быть способны составить непротиворечивое множество аксиом для получения ответов на поставленный здесь ограниченный круг вопросов.

- a)** Джон — ребенок или взрослый? [Взрослый]
  - б)** У Джона теперь имеется не меньше двух помидоров? [Да]
  - в)** Купил ли Джон какие-либо мясные продукты? [Да]
  - г)** Если бы Мэри покупала помидоры в то же время, что и Джон, увидел бы он ее? [Да]
  - д)** Сделаны ли помидоры в супермаркете? [Нет]
  - е)** Что Джон собирается сделать с помидорами? [Съесть их]
  - ж)** В супермаркете Safeway продаются дезодоранты? [Да]
  - з)** Джон принес какие-то деньги в супермаркет? [Да]
  - и)** Стал ли Джон иметь меньше денег после поездки в супермаркет? [Да]
- 10.23.** Внесите необходимые дополнения или изменения в вашу базу знаний, разработанную в предыдущем упражнении, так, чтобы можно было отвечать на вопросы, которые следуют из этих знаний. Покажите, что на них действительно можно найти ответы с помощью этой базы знаний, и включите в ваш отчет описание исправлений, объяснение того, почему они потребовались, указание, являются ли эти исправления мелкими или крупными, и т.д.

- a) Находились ли другие люди в супермаркете Safeway в то время, как там был Джон? [Да — работники супермаркета!]
  - б) Является ли Джон вегетарианцем? [Нет]
  - в) Кому принадлежит дезодорант в супермаркете Safeway? [Корпорации Safeway]
  - г) Имеется ли у Джона унция говяжьего фарша? [Да]
  - д) Имеется ли бензин в расположенной рядом бензозаправочной станции Shell? [Да]
  - е) Имеются ли помидоры в багажнике автомобиля Джона? [Да]
- 10.24.** Напомним, что информация о наследовании в семантических сетях может быть представлена логически с помощью подходящих импликационных высказываний. В данном упражнении рассматривается эффективность использования подобных высказываний для представления информации о наследовании.
- a) Рассмотрим содержание каталога подержанных автомобилей, такого как *Kelly's Blue Book*, например, сведения о том, что микроавтобусы типа 1973 Dodge Van стоят 575 долларов. Предположим, что вся эта информация (об 11 тысячах моделей) закодирована в виде логических правил, как было предложено в данной главе. Запишите три таких правила, включая правило, касающееся микроавтобусов типа 1973 Dodge Van. Как вы сможете воспользоваться этими правилами, чтобы найти стоимость конкретного автомобиля (например, модели JV, которая относится к типу 1973 Dodge Van) при наличии такого средства автоматического доказательства теорем по методу обратного логического вывода, как Prolog?
  - б) Сравните временную эффективность использования для решения этой задачи метода обратного логического вывода и метода наследования, используемого в семантических сетях.
  - в) Объясните, как метод прямого логического вывода позволяет использовать систему на основе логики для эффективного решения той же задачи, при условии, что база знаний содержит только 11 тысяч правил, касающихся цен.
  - г) Опишите ситуацию, в который ни прямой, ни обратный логический вывод с помощью этих правил не позволяет эффективно выполнить запрос о цене на какой-то отдельный автомобиль.
  - д) Можете ли вы предложить решение, позволяющее эффективно отвечать на запросы этого типа во всех вариантах логических систем? (*Подсказка.* Напомним, что два автомобиля одной и той же категории имеют одинаковую цену.)
- 10.25.** Можно было бы предположить, что не требуется подчеркивать в семантических сетях синтаксические различия между связями с метками, не заключенными в прямоугольники, и связями с метками в одинарном прямоугольнике, поскольку связи с метками в одинарном прямоугольнике всегда закреплены за категориями; а в алгоритме наследования, допустим, можно было бы просто принять допущение, что связь с меткой без прямоугольника, закрепленная за некоторой категорией, предназначена для применения ко всем элементам этой категории. Покажите, что такое утверждение является ошибочным, и приведите примеры ошибок, которые могут возникнуть из-за этого.



## Часть IV

# ПЛАНИРОВАНИЕ

Основы планирования	512
Планирование и осуществление действий в реальном мире	564

# 11 ОСНОВЫ ПЛАНИРОВАНИЯ

*В данной главе показано, каким образом агент может воспользоваться информацией о структуре задачи для создания сложных планов действий.*

**Планированием** называется процесс выработки последовательности действий, позволяющих достичь цели. До сих пор в этой книге рассматривались два примера планирующих агентов: описанный в главе 3 агент, осуществляющий решение задач на основе поиска, и логический планирующий агент, который представлен в главе 10. Данная глава в основном посвящена описанию вопросов расширения области применения агентов и ее распространения на сложные задачи планирования, которые нельзя решить с помощью подходов, рассматривавшихся до сих пор.

В разделе 11.1 разрабатывается выразительный, но тщательно регламентированный язык для представления задач планирования, включая действия и состояния. Этот язык тесно связан с представлениями действий в пропозициональной логике и логике первого порядка, которые рассматривались в главах 7 и 10. В разделе 11.2 описано, как можно использовать эти представления в алгоритмах прямого и обратного поиска в основном с применением точных эвристик, которые могут создаваться автоматически исходя из структуры представления. (Используемый при этом способ аналогичен способу формирования эффективных эвристик для задач удовлетворения ограничений в главе 5.) В разделах 11.3–11.5 описаны алгоритмы планирования, которые выходят за рамки прямого и обратного поиска и позволяют воспользоваться имеющимся представлением задачи. В частности, представлены подходы, позволяющие не ограничиваться рассмотрением только полностью упорядоченных последовательностей действий.

В данной главе рассматриваются лишь такие варианты среды, которые являются полностью наблюдаемыми, детерминированными, конечными, статическими (изменения происходят только в результате действий агента) и дискретными (с точки зрения времени, действий, объектов и результатов). Такая среда называется средой **классического планирования**. В отличие от этого, неклассическое планирование предназначено для частично наблюдаемых или стохастических вариантов среды и для него требуется другой набор алгоритмов и проектов агентов, описанный в главах 12 и 17.

## 11.1. ЗАДАЧА ПЛАНИРОВАНИЯ

Рассмотрим, что может произойти, когда обычный агент, решающий задачи с помощью стандартных алгоритмов поиска (поиска в глубину, поиска A\* и т.д.), сталкивается с крупными задачами реального мира. Это позволит нам научиться разрабатывать лучших планирующих агентов.

Наиболее очевидная сложность состоит в том, что агент, решающий задачи, может быть просто подавлен огромным количеством действий, не относящихся к делу. Рассмотрим задачу покупки одного экземпляра англоязычного издания настоящей книги с названием *AI: A Modern Approach* в электронном книжном магазине. Предположим, что агент-покупатель должен совершить одно действие, связанное с покупкой, в расчёте на каждый возможный десятицифровой номер ISBN, что приводит к общему количеству действий, равному 10 миллиардам. В ходе применения алгоритма поиска агент должен исследовать состояния результатов всех 10 миллиардов действий, чтобы определить, какое из них соответствует цели, заключающейся в том, чтобы приобрести экземпляр книги с номером ISBN 0137903952. С другой стороны, разумный планирующий агент должен быть способным проработать процедуру покупки в обратном направлении, от явного описания цели, такого как *Have(ISBN0137903952)*, и непосредственно сформировать действие *Buy(ISBN0137903952)*. Для этого агенту требуется иметь общие знания о том, что действие *Buy(x)* приводит к результату *Have(x)*. При наличии этих знаний и цели планировщик может определить в единственном шаге унификации, что правильным действием является *Buy(ISBN0137903952)*.

Еще одна сложность заключается в определении хорошей **эвристической функции**. Предположим, что цель агента состоит в том, чтобы купить четыре разных книги в оперативном режиме. Количество планов только для четырех этапов покупки будет составлять  $10^{40}$ , поэтому поиск без точной эвристики даже нет смысла рассматривать. Для человека очевидно, что хорошей эвристической оценкой для стоимости состояния является количество книг, которые еще предстоит купить; к сожалению, эта идея не столь очевидна для агента, решающего задачи, поскольку он рассматривает процедуру проверки цели как “черный ящик”, который возвращает истину или ложь в ответ на каждое состояние. Поэтому агент, решающий задачи, не обладает автономностью; он требует, чтобы человек предоставлял ему эвристическую функцию для каждой новой задачи. С другой стороны, если планирующий агент имеет доступ к явному представлению цели как конъюнкции подцелей, то может использовать единственную эвристику, независимую от проблемной области, — количество невыполненных конъюнктоов. Для задачи покупки книг цель будет представлять собой выражение *Have(A)  $\wedge$  Have(B)  $\wedge$  Have(C)  $\wedge$  Have(D)*, а состояние, содержащее выражение *Have(A)  $\wedge$  Have(C)*, будет иметь стоимость 2. Таким образом, агент автоматически получает правильную эвристику для этой задачи и для многих других. Ниже в этой главе будет показано, как формировать более сложные эвристики, в которых учитываются не только структура цели, но и возможные действия.

Наконец, агент-решатель задач может оказаться неэффективным из-за того, что не способен воспользоваться  **декомпозицией задачи**. Рассмотрим задачу доставки множества пакетов почты по соответствующим адресатам, которые разбросаны по всей Австралии. В данном случае имеет смысл найти аэропорт, ближайший к каждому адресату, и разделить общую задачу на несколько подзадач, по одной на

каждый аэропорт. А в самом множестве пакетов, доставляемых через каждый конкретный аэропорт, можно определить в зависимости от города адресата допустимость дальнейшей декомпозиции. Как было описано в главе 5, способность выполнять декомпозицию такого рода обеспечивает повышение эффективности решателей задач удовлетворения ограничений. Такое же утверждение остается справедливым для планировщиков: в наихудшем случае может потребоваться время  $O(n!)$  для поиска наилучшего плана доставки  $n$  пакетов, но если существует возможность выполнить декомпозицию этой задачи на  $k$  равных частей, затраты времени будут составлять только  $O((n/k)! \times k)$ .

Как было отмечено в главе 5, идеально декомпонуемые задачи являются очень привлекательными, но встречаются редко<sup>1</sup>. Проекты многих систем планирования (особенно планировщиков с частичным упорядочением, описанных в разделе 11.3) основаны на предположении, что большинство задач реального мира являются ~~частично декомпонуемыми~~. Это означает, что планировщик может работать над подцелями независимо, но, скорее всего, ему потребуется также выполнить определенную дополнительную работу по объединению результирующих субпланов. Применительно к некоторым задачам такое предположение не оправдывается, поскольку велика вероятность того, что работа над одной подцелью будет приводить к разрушению результатов работы над другой подцелью. Именно из-за такого взаимодействия подцелей и являются трудноразрешимыми многие головоломки (подобные задаче игры в восемь).

## Язык задач планирования

Как следует из приведенного выше описания, сам способ представления задач планирования (состояний, действий и целей) должен обеспечивать возможность для алгоритмов планирования воспользоваться логической структурой задачи. Весь секрет состоит в том, чтобы найти язык, достаточно выразительный для описания широкого круга задач, но вместе с тем достаточно ограничительный для обеспечения функционирования на его основе эффективных алгоритмов. В этом разделе вначале рассматривается основной язык представления классических планировщиков, известный под названием<sup>2</sup> Strips. Затем кратко представлены некоторые из многих возможных вариантов языков, подобных Strips.

- **Представление состояний.** В планировщиках применяется декомпозиция мира на логические условия, а состояние представляется в виде конъюнкции положительных литералов. Мы будем рассматривать пропозициональные литералы; например, выражение *Poor*  $\wedge$  *Unknown* может представлять состояние агента-неудачника (бедного и безвестного). В этой главе будут также использоваться литералы первого порядка; например, выражение *At(Plane<sub>1</sub>, Melbourne)*  $\wedge$  *At(Plane<sub>2</sub>, Sydney)* может представлять одно из состояний в задаче доставки

<sup>1</sup> Следует отметить, что даже задача доставки пакетов не является идеально декомпонуемой. Могут возникать такие случаи, в которых лучше назначить пакеты для доставки в более отдаленный аэропорт, если это позволит исключить необходимость еще одного полета в более близкий аэропорт. Тем не менее большинство компаний, занимающихся доставкой почты, предпочитают решения, характеризующиеся вычислительной и организационной простотой, таким решениям, которые продиктованы требованиями декомпозиции.

<sup>2</sup> Strips — сокращение от STanford Research Institute Problem Solver.

пакетов. Литералы в описаниях состояния первого порядка должны быть **базовыми и не содержащими функций**. Такие литералы, как  $At(x, y)$  или  $At(Father(Fred), Sydney)$ , не допускаются. Кроме того, используется **предположение о замкнутом мире**, которое означает, что любые условия, не упомянутые в описании состояния, считаются ложными.

- **Представление целей.** Цель — это частично заданное состояние, представленное в виде конъюнкции положительных базовых литералов, таких как  $Rich \wedge Famous$  или  $At(P_2, Tahiti)$ . Пропозициональное состояние  $s$   $\bowtie$  **удовлетворяет** цели  $g$ , если  $s$  содержит все атомы из  $g$  (и, возможно, другие атомы). Например, состояние  $Rich \wedge Famous \wedge Miserable$  (богатый, знаменитый и несчастный) удовлетворяет цели  $Rich \wedge Famous$ .
- **Представление действий.** Любое действие задается в терминах предусловий, которые должны соблюдаться до того, как оно может быть выполнено, и результатов, которые становятся следствием его выполнения. Например, действие, состоящее в перелете самолета из одного места в другое, можно записать следующим образом:

```
Action(Fly(p, from, to),
  Precond: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to),
  Effect: ¬At(p, from) ∧ At(p, to))
```

где **Precond** обозначает предусловие, а **Effect** — результат.

Такое выражение следовало бы называть не просто действием, а  $\bowtie$  **схемой действия**, в том смысле, что в нем представлен целый ряд различных действий, которые могут явиться следствием конкретизации переменных  $p$ ,  $from$  и  $to$  различными константами. Вообще говоря, любая схема действия состоит из перечисленных ниже трех частей.

- Имя действия и список параметров (например,  $Fly(p, from, to)$ ) служат для обозначения действия.
- $\bowtie$  **Предусловием** называется конъюнкция не содержащих функций положительных литералов, регламентирующая то, что должно быть истинным в некотором состоянии, прежде чем можно будет выполнить рассматриваемое действие. Любые переменные, заданные в предусловии, должны также присутствовать в списке параметров действия.
- $\bowtie$  **Результатом** является конъюнкция не содержащих функций литералов, представляющая собой описание того, как изменится состояние после выполнения действия. В состоянии, ставшем результатом действия, подтверждается истинность положительного литерала  $P$  в результате, тогда как применительно к отрицательному литералу  $\neg P$  подтверждается его ложность. Переменные, присутствующие в результате, должны также присутствовать в списке параметров действия.

Для повышения удобства чтения в некоторых системах планирования предусмотрено разделение результата на  $\bowtie$  **список добавления** для положительных литералов и  $\bowtie$  **список удаления** для отрицательных литералов.

Определив синтаксис представлений задач планирования, можно приступить к определению семантики. Наиболее простой способ выполнения этой задачи со-

стоит в том, чтобы описать, как действия влияют на состояния. (Альтернативный метод предусматривает определение непосредственного преобразования в аксиомы состояния-преемника, семантика которых основана на логике первого порядка; см. упр. 11.3.) Вначале необходимо отметить, что действие **применимо** в любом состоянии, в котором выполняется предусловие; в противном случае действие не имеет эффекта. Для схемы действий первого порядка задача определения применимости связана с поиском подстановки  $\theta$  для переменных в предусловии. Например, предположим, что текущее состояние описано следующим образом:

$$\begin{aligned} & At(P_1, JFK) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \\ & \quad \wedge Airport(JFK) \wedge Airport(SFO) \end{aligned}$$

Это состояние удовлетворяет такому предусловию:

$$At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$

с подстановкой  $\{p/P_1, from/JFK, to/SFO\}$  (кроме всего прочего; см. упр. 11.2). Поэтому конкретное действие  $Fly(P_1, JFK, SFO)$  является применимым.

Начиная с состояния  $s$ , **результатом** выполнения применимого действия  $a$  является состояние  $s'$ , представляющее собой то же самое, что и  $s$ , за исключением того, что любой положительный литерал  $P$  в результате  $a$  добавляется к  $s'$ , а любой отрицательный литерал  $\neg P$  удаляется из  $s'$ . Таким образом, после выполнения действия  $Fly(P_1, JFK, SFO)$  показанное выше текущее состояние принимает вид

$$\begin{aligned} & At(P_1, SFO) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \\ & \quad \wedge Airport(JFK) \wedge Airport(SFO) \end{aligned}$$

Следует отметить, что если какой-то положительный результат уже находится в состоянии  $s$ , то не добавляется повторно, а если какой-то отрицательный результат отсутствует в состоянии  $s$ , эта часть результата игнорируется. Такое определение воплощает в себе так называемое **предположение Strips**, что каждый литерал, не упомянутый в результате, остается неизменным. Благодаря этому язык Strips позволяет избежать возникновения проблемы **представительного окружения**, описанной в главе 10.

Наконец, можно определить **решение** для задачи планирования. В своей простейшей форме это всего лишь последовательность действий, которая, будучи выполненной в начальном состоянии, приводит к состоянию, удовлетворяющему цели. Далее в этой главе будет показано, как обеспечить возможность использовать решение, представляющее собой частично упорядоченные множества действий, при условии, что каждая последовательность действий, которая отвечает этому частичному упорядочению, является решением.

## Выразительность и расширения языка

Всевозможные ограничения, налагаемые в представлении Strips, были выбраны в надежде на то, что алгоритмы планирования станут более простыми и эффективными и вместе с тем при описании реальных задач не будут возникать слишком значительные трудности. Одно из самых важных ограничений состоит в том, что литералы должны быть свободными от функций. Благодаря наличию такого ограничения можно быть уверенным в том, что каждую схему действий для каждой конкретной задачи удастся пропозиционализировать, т.е. преобразовать в конечную коллекцию чисто пропозициональных представлений действий без переменных (дополнительную информа-

цию по этой теме см. в главе 9). Например, в проблемной области грузовых авиационных перевозок для задачи с десятью самолетами и пятью аэропортами можно преобразовать схему  $Fly(p, from, to)$  в  $10 \times 5 \times 5 = 250$  чисто пропозициональных действий. Планировщики, описанные в разделах 11.4 и 11.5, работают непосредственно с пропозиционализированными описаниями. Если же будут разрешены функциональные символы, то количество состояний и действий, которые могут быть сконструированы, станет бесконечно большим.

В последние годы стало очевидно, что язык Strips является недостаточно выразительным для некоторых реальных проблемных областей. В результате этого было разработано много вариантов данного языка. В табл. 11.1 кратко описан наиболее важный из них, язык **ADL** (Action Description Language — язык описания действий), на основе его сравнения с базовым языком Strips. В языке ADL действие  $Fly$  можно записать следующим образом:

```
Action(Fly(p:Plane, from:Airport, to:Airport),
  Precond: At(p, from) ∧ (from ≠ to),
  Effect: ¬At(p, from) ∧ At(p, to))
```

**Таблица 11.1. Сравнение языков представления задач планирования Strips и ADL. В обоих случаях цели выглядят как предусловия некоторого действия без параметров**

Язык Strips	Язык ADL
Только положительные литералы в состояниях: $Poor \wedge Unknown$	Положительные и отрицательные литералы в состояниях: $\neg Rich \wedge \neg Famous$
Предположение о замкнутом мире: неупомянутые литералы являются ложными	Предположение об открытом мире: неупомянутые литералы являются неизвестными
Результат $P \wedge \neg Q$ означает добавление $P$ и удаление $Q$	Результат $P \wedge \neg Q$ означает добавление $P$ и $\neg Q$ и удаление $\neg P \wedge Q$
Только базовые литералы в целях: $Rich \wedge Famous$	Квантифицированные переменные в целях: цель $\exists x At(P_1, x) \wedge At(P_2, x)$ состоит в том, чтобы $P_1$ и $P_2$ находились в одном и том же месте
Цели представляют собой конъюнкции: $Rich \wedge Famous$	В целях допускается использование конъюнкций и дизъюнкций: $\neg Poor \wedge (Famous \vee Smart)$
Результаты представляют собой конъюнкции	Разрешены условные результаты: <b>when</b> $P$ : $E$ означает, что $E$ является результатом, только если выполнено условие $P$
Отсутствует поддержка отношения равенства	Предикат равенства ( $x=y$ ) является встроенным
Отсутствует поддержка типов	Переменные могут иметь типы, как, например, в выражении $(p: Plane)$

Обозначение  $p: Plane$  в списке параметров представляет собой сокращение для  $Plane(p)$  в предусловии; применение такого обозначения не приводит к повышению выразительной мощи, но позволяет повысить удобство для чтения. (Применение этого обозначения приводит также к сокращению количества возможных пропозициональных действий, которые могут быть сконструированы.) Предусловие ( $from \neq to$ ) выражает тот факт, что полет не может быть совершен из некоторого аэропорта в сам этот аэропорт. Такое условие не может быть выражено кратко в языке Strips.

Различные формальные средства планирования, применяемые в искусственном интеллекте, были систематизированы в рамках стандартного синтаксиса, получившего название PDDL (Planning Domain Definition Language — язык определения проблемной области планирования). Этот язык позволяет исследователям обмениваться эталонными тестовыми задачами и сравнивать полученные результаты. Язык PDDL включает подязыки для Strips, ADL и иерархических сетей задач (Hierarchical Task Network — HTN), которые будут рассматриваться в главе 12.

Системы обозначений Strips и ADL являются вполне приемлемыми для многих реальных проблемных областей. В приведенных ниже подразделах описаны некоторые простые примеры. Тем не менее эти языки все еще характеризуются некоторыми существенными ограничениями. Наиболее очевидным из них является то, что эти языки не позволяют представить естественным образом **последствия** действий. Например, если в самолете есть люди, пакеты или частички пыли, то все они в процессе полета самолета также меняют свое местонахождение. Можно было бы представить эти изменения как прямой результат полета, но кажется более естественным представить изменение местонахождения содержимого самолета как логическое следствие изменения местонахождения самого самолета. Дополнительные примеры подобных ~~и~~ ограничений состояния приводятся в разделе 11.5. Кроме того, в классических системах планирования не предпринимается даже попытка решить проблему **спецификации**: проблему не представленных в определении задачи обстоятельств, которые могут вызвать неудачное завершение действия. Способы решения проблемы спецификации описаны в главе 12.

### Пример: воздушный грузовой транспорт

В листинге 11.1 показана одна из задач организации перевозок с помощью воздушного грузового транспорта, в которой предусматривается загрузка и разгрузка грузов в самолеты и из самолетов, а также перелет из одного места в другое. Эта задача может быть определена с помощью трех действий: *Load*, *Unload* и *Fly*. Действия влияют на значения двух предикатов: предикат *In*(*c*, *p*) означает, что груз *c* находится в самолете *p*, а предикат *At*(*x*, *a*) означает, что объект *x* (либо самолет, либо груз) находится в аэропорту *a*. Следует отметить, что предикат *At* больше вообще не обуславливает местонахождение груза, если предикат *In* определяет, что груз находится в самолете, поэтому *At* фактически означает “доступен для использования в указанном местонахождении”. Для того чтобы научиться прорабатывать все эти детали должным образом, необходимо приобрести определенный опыт составления определений действий. Ниже приведен план, который является решением данной задачи.

```
[Load(C1, P1, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK),
 Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO)]
```

#### Листинг 11.1. Задача Strips, в которой предусматривается транспортировка груза между аэропортами

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK)
     ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2)
     ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))

Action(Load(c, p, a),
       ...)
```

---

```

Precond: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
Effect: ¬At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
      Precond: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
      Effect: At(c, a) ∧ ¬In(c, p))
Action(Fly(p, from, to),
      Precond: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
      Effect: ¬At(p, from) ∧ At(p, to))

```

---

В применяемом здесь представлении используется чистый (т.е. не дополненный) язык Strips. В частности, это представление не исключает того, что самолет может вылететь и прилететь в один и тот же аэропорт. Возникновение такой ситуации могут предотвратить литералы неравенства, предусмотренные в языке ADL.

### Пример: задача с запасным колесом

Рассмотрим задачу смены колеса со стертой покрышкой. Точнее, цель состоит в том, чтобы на оси автомобиля было правильно смонтировано исправное запасное колесо, тогда как в начальном состоянии на оси имеется колесо со стертой покрышкой, а в багажнике находится исправное запасное колесо. Для того чтобы упростить эту задачу, рассмотрим чрезвычайно абстрактную ее версию, в которой не учитываются крепко прихваченные крепежные гайки или другие усложнения. Существуют только четыре действия: выемка запасного колеса из багажника, снятие колеса со стертой покрышкой с оси, установка запасного колеса на ось и оставление автомобиля без присмотра на ночь. Предполагается, что автомобиль находится в районе с исключительно неблагоприятной криминогенной обстановкой, поэтому результатом его пребывания ночью на улице становится исчезновение колес.

Описание ADL этой задачи приведено в листинге 11.2. Обратите внимание на то, что оно является исключительно пропозициональным. Это описание превосходит по своим возможностям описание на языке Strips в том, что в его предусловии для действия *PutOn(Spare, Axle)* (поместить запасное колесо на ось) используется отрицаемый предикат *¬At(Flat, Axle)* (отрицание предиката *At(Flat, Axle)* — на оси находится колесо со стертой покрышкой). Необходимости в этом можно избежать, применяя вместо него предикат *Clear(Axle)* (отсутствие колеса на оси), как будет показано в следующем примере.

### Листинг 11.2. Простая задача с запасным колесом

---

```

Init(At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk),
       Precond: At(Spare, Trunk)
       Effect: ¬At(Spare, Trunk) ∧ At(Spare, Ground))
Action(Remove(Flat, Axle),
       Precond: At(Flat, Axle)
       Effect: ¬At(Flat, Axle) ∧ At(Flat, Ground))
Action(PutOn(Spare, Axle),
       Precond: At(Spare, Ground) ∧ ¬At(Flat, Axle))

```

---

```

Effect:  $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \text{At}(\text{Spare}, \text{Axle})$ 
Action(LeaveOvernight,
Precond:
Effect:  $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \neg \text{At}(\text{Spare}, \text{Axle}) \wedge \neg \text{At}(\text{Spare}, \text{Trunk})$ 
 $\wedge \neg \text{At}(\text{Flat}, \text{Ground}) \wedge \neg \text{At}(\text{Flat}, \text{Axle})$ )

```

---

### Пример: мир блоков

Одна из наиболее широко известных проблемных областей планирования известна под названием **мира блоков**. Эта проблемная область состоит из множества блоков кубической формы, находящихся на столе<sup>3</sup>. Блоки можно укладывать в столбик, но на верхней поверхности одного блока может быть непосредственно размещен только еще один блок. Робот может брать манипулятором блок и перемещать его в другую позицию — либо на стол, либо на верхнюю поверхность другого блока. С помощью манипулятора может быть взят одновременно только один блок, поэтому невозможно взять блок, на котором стоит еще один блок. Цель всегда заключается в том, что должны быть построены один или несколько столбиков из блоков, а сами задачи формулируются в терминах того, какие блоки находятся над указанными другими блоками. Например, задача может состоять в том, чтобы поставить блок A на B и блок C на D.

Для указания на то, что блок  $b$  находится на блоке  $x$ , где  $x$  — либо другой блок, либо стол, используется предикат  $\text{On}(b, x)$ . Для перемещения блока  $b$  с верхней поверхности блока  $x$  на верхнюю поверхность блока  $y$  применяется действие  $\text{Move}(b, x, y)$ . Итак, одним из предусловий перемещения  $b$  является то, что на нем не стоит какой-то другой блок. В логике первого порядка оно может быть представлено с помощью выражения  $\neg \exists x \text{ On}(x, b)$  или альтернативного выражения  $\forall x \neg \text{On}(x, b)$ . Эти выражения могут быть сформулированы как предусловия на языке ADL. Но мы могли бы оставаться в рамках языка Strips, введя новый предикат,  $\text{Clear}(x)$ , который является истинным, если на блоке  $x$  ничего нет (верхняя поверхность блока  $x$  свободна).

Действие  $\text{Move}$  позволяет переместить блок  $b$  с блока  $x$  на блок  $y$ , если свободны верхние поверхности и блока  $b$ , и блока  $y$ . После выполнения этого действия верхняя поверхность блока  $x$  свободна, а блока  $y$  — нет. Формальное описание действия  $\text{Move}$  в языке Strips состоит в следующем:

```

Action(LeaveOvernight,
Precond:  $\text{On}(b, \text{Ground}) \wedge \text{Clear}(b) \wedge \text{Clear}(y)$ ,
Effect:  $\text{On}(b, y) \wedge \text{Clear}(x) \wedge \neg \text{On}(b, x) \wedge \neg \text{Clear}(y)$ )

```

К сожалению, в этом действии предикат  $\text{Clear}$  не сопровождается должным образом, если блок  $x$  или блок  $y$  находится на столе. Если  $x=\text{Table}$ , то результатом этого действия будет  $\text{Clear}(\text{Table})$ , но поверхность стола не должна становиться свободной (поскольку она и так всегда свободна), а если  $y=\text{Table}$ , то действие имеет предусловие  $\text{Clear}(\text{Table})$ , но вся поверхность стола не обязана быть свободной для того, чтобы на нее можно было поместить блок (поскольку на столе всегда и

<sup>3</sup> Мир блоков, используемый в этих исследованиях планирования, гораздо проще по сравнению с версией Shrdlu, которая показана на с. 60.

без того достаточно места). Для исправления такого положения необходимо предусмотреть два изменения. Во-первых, введем еще одно действие для перемещения блока  $b$  с блока  $x$  на стол:

```
Action(MoveToTable(b, x),
  Precond: On(b, x) ∧ Clear(b),
  Effect: On(b, Table) ∧ Clear(x) ∧ ¬On(b, x))
```

Во-вторых, примем интерпретацию предиката  $Clear(b)$  как означающую, что “на  $b$  есть достаточно свободного места, чтобы поместился один блок”. При этой интерпретации предикат  $Clear(Table)$  всегда будет истинным. Единственная проблема состоит в том, что ничто не будет препятствовать планировщику использовать действие  $Move(b, x, Table)$  вместо действия  $MoveToTable(b, x)$ . Можно смириться с этой проблемой (она приводит к созданию пространства поиска с размерами больше необходимого, но не становится причиной получения неправильных ответов) или ввести в предусловие  $Move$  предикат  $Block$  и добавить в предусловие действия  $Move$  выражение  $Block(b) \wedge Block(y)$ .

Наконец, существует проблема фиктивных действий, таких как  $Move(B, C, C)$ , которые должны представлять собой пустую операцию, но имеют противоречивые результаты. Обычно принято игнорировать подобные проблемы, поскольку они редко вызывают выработку неверных планов. Правильный подход состоит в добавлении предусловий в неравенство, как показано в листинге 11.3.

**Листинг 11.3. Задача планирования в мире блоков: построение столбика из трех блоков. Одним из решений является последовательность действий [ $Move(B, Table, C), Move(A, Table, B)$ ]**

---

```
Init(On(A, Table) ∧ On(B, Table) ∧ On(C, Table)
     ∧ Block(A) ∧ Block(B) ∧ Block(C)
     ∧ Clear(A) ∧ Clear(B) ∧ Clear(C))
Goal(On(A, B) ∧ On(B, C))
Action(Move(b, x, y),
  Precond: On(b, x) ∧ Clear(b) ∧ Clear(y) ∧ Block(b) ∧
            (b ≠ x) ∧ (b ≠ y) ∧ (x ≠ y),
  Effect: On(b, y) ∧ Clear(x) ∧ ¬On(b, x) ∧ ¬Clear(y))
Action(MoveToTable(b, x),
  Precond: On(b, x) ∧ Clear(b) ∧ Block(b) ∧ (b ≠ x),
  Effect: On(b, Table) ∧ Clear(x) ∧ ¬On(b, x))
```

---

## 11.2. ПЛАНИРОВАНИЕ С ПОМОЩЬЮ ПОИСКА В ПРОСТРАНСТВЕ СОСТОЯНИЙ

Данный раздел посвящен описанию алгоритмов планирования. Наиболее простой подход состоит в использовании поиска в пространстве состояний. Поскольку описания действий в задаче планирования определяют и предусловия, и результаты, существует возможность организовать поиск в обоих направлениях: либо в прямом, от начального состояния, либо в обратном, от цели, как показано на рис. 11.1. Кроме того, явные представления действий и целей могут использоваться для автоматического вывода эффективных эвристик.

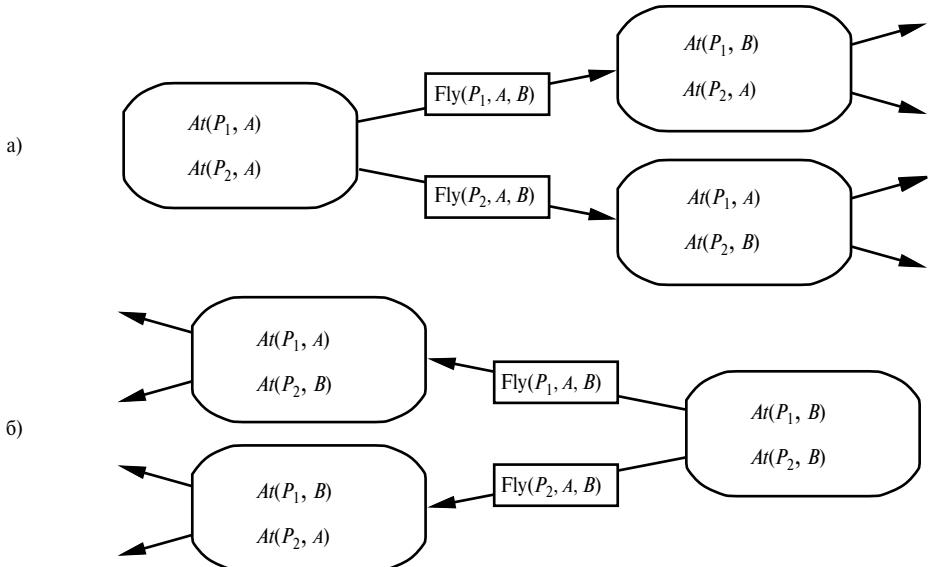


Рис. 11.1. Два подхода к организации поиска плана: прямой (прогрессивный) поиск в пространстве состояний, начинающийся с начального состояния, в котором используются действия задачи для прямого поиска целевого состояния (а); обратный (ретрессивный) поиск в пространстве состояний: поиск доверительного состояния (см. с. 141), начинающийся с целевого состояния (состояний), в котором для обратного поиска начального состояния используются инверсии действий (б)

### Прямой поиск в пространстве состояний

Подход к организации планирования с использованием прямого поиска в пространстве состояний аналогичен подходу к решению задач, описанному в главе 3. Иногда его называют **прогрессивным** планированием, поскольку оно предусматривает продвижение в прямом направлении. При этом мы начинаем с начального состояния задачи и рассматриваем последовательности действий до тех пор, пока не находим последовательность, позволяющую достичь целевого состояния. Формулировка задач планирования в виде задач поиска в пространстве состояний приведена ниже.

- **Начальным состоянием** поиска является начальное состояние, взятое из задачи планирования. Вообще говоря, каждое состояние должно представлять собой множество положительных базовых литералов; литералы, не присутствующие в определении задачи, являются ложными.
- Всеми **действиями**, применимыми в некотором состоянии, являются такие действия, для которых выполнены предусловия. Состояние-преемник, являющееся результатом выполнения действия, вырабатывается путем добавления положительных литералов результата и удаления отрицательных литералов результата. (В случае использования литералов первого порядка к литералам результата необходимо применять унификатор из предусловий.) Обратите внимание на то, что во всех задачах планирования используется единственная функция определения преемника, что является следствием применения явного представления действий.

- В ходе **проверки цели** осуществляется проверка того, удовлетворяет ли данное состояние цели условиям задачи планирования.
- **Стоимость этапа** для каждого действия обычно равна 1. Хотя было бы несложно предусмотреть использование различных стоимостей для разных действий, такая возможность в планировщиках Strips применяется редко.

Напомним, что при отсутствии функциональных символов пространство состояний задачи планирования является конечным. Поэтому полным алгоритмом планирования должен быть любой алгоритм поиска в графе, который является полным (например, A\*).

Начиная с самых первых дней исследований по планированию (примерно с 1961 года) и до сравнительно недавнего времени (примерно до 1998 года), предлагалось, что прямой поиск в пространстве состояний был бы слишком неэффективен для того, чтобы иметь практическое значение. Причины этого понять совсем не сложно — достаточно вернуться в начало раздела 11.1. Во-первых, прямой поиск не решает проблему не относящихся к делу действий — рассматриваются все действия, применимые из каждого состояния. Во-вторых, без хорошей эвристики этот подход быстро приводит к чрезмерному увеличению объема вычислений. Рассмотрим задачу воздушных грузовых перевозок с 10 аэропортами, где в каждом аэропорту имеется 5 самолетов и 20 единиц груза. Цель состоит в том, чтобы переместить все грузы из аэропорта *A* в аэропорт *B*. Существует простое решение этой задачи: погрузить 20 единиц груза в один из самолетов в аэропорту *A*, полететь на этом самолете в аэропорт *B* и выгрузить груз. Но формальный поиск этого решения может оказаться затруднительным, поскольку средний коэффициент ветвления является огромным: каждый из этих  $5 \times 10 = 50$  самолетов может полететь в 9 других аэропортов, а каждая из  $20 \times 10 = 200$  единиц груза может быть либо разгружена (если она загружена), либо загружена в любой самолет в аэропорту, где она находится (если она разгружена). Допустим, что в среднем существует около 1000 возможных действий, поэтому дерево поиска с глубиной, равной глубине очевидного решения, имеет около 1000<sup>41</sup> узлов. Очевидно, что для обеспечения эффективности такого поиска потребуется очень точная эвристика. Мы рассмотрим некоторые возможные эвристики после описания обратного поиска.

## Обратный поиск в пространстве состояний

Обратный поиск в пространстве состояний был кратко описан в составе двунаправленного поиска в главе 3. В этой главе было отмечено, что задача организации обратного поиска может оказаться сложной, если целевые состояния описаны с помощью множества ограничений, а не перечислены явно. В частности, не всегда очевидно, как должно составляться описание возможных **преемников** множества целевых состояний. А в этой главе будет показано, что представление Strips позволяет решить такую задачу очень легко, поскольку множества состояний могут быть описаны с помощью литералов, которые должны быть истинными в этих состояниях.

Основным преимуществом обратного поиска является то, что он позволяет рассматривать только **релевантные** действия. Действие релевантно конъюнктивной цели, если оно достигает одного из конъюнктов данной цели. Например, целью опи-

санной выше задачи по воздушной перевозке грузов с 10 аэропортами была доставка 20 единиц груза в аэропорт  $B$  или, точнее:

$$At(C_1, B) \wedge At(C_2, B) \wedge \dots \wedge At(C_{20}, B)$$

Теперь рассмотрим конъюнкту  $At(C_1, B)$ . Двигаясь в обратном направлении, можно найти действия, имеющие этот результат. Таковым является только одно действие:  $Unload(C_1, p, B)$ , где самолет  $p$  не задан.

Обратите внимание на то, что имеется также много нерелевантных действий, способных привести к целевому состоянию. Например, можно организовать перелет пустого самолета из аэропорта  $JFK$  в аэропорт  $SFO$ ; это действие позволяет достичь целевого состояния из состояния-предшественника, в котором самолет находился в  $JFK$  и все целевые конъюнкты были удовлетворены. Обратный поиск, в котором допускаются нерелевантные действия, по-прежнему будет полным, но гораздо менее эффективным. Если решение существует, то должно быть найдено с помощью обратного поиска, допускающего только релевантные действия. Наличие такого ограничения, в котором допускаются только релевантные действия, означает, что процедура обратного поиска часто имеет гораздо более низкий коэффициент ветвления по сравнению с прямым поиском. Например, в рассматриваемой задаче грузовых воздушных перевозок имеется около 1000 действий, ведущих в прямом направлении из начального состояния, но только 20 действий, позволяющих перейти в обратном направлении от цели.

Поиск в обратном направлении иногда называют **регрессивным** планированием. Основной вопрос регрессивного планирования состоит в следующем: есть ли такие состояния, из которых применение некоторого действия приводит к цели? Вычисление описаний таких состояний называется **регрессией** цели через действие. Для того чтобы определить, как можно найти ответ на указанный выше вопрос, рассмотрим пример с воздушными грузовыми перевозками. Мы имеем цель

$$At(C_1, B) \wedge At(C_2, B) \wedge \dots \wedge At(C_{20}, B)$$

и релевантное действие  $Unload(C_1, p, B)$ , которое позволяет достичь первого конъюнкта. Соответствующее действие будет выполнимо, только если выполнены его предусловия. Поэтому любое состояние-преемник должно включать эти предусловия:  $In(C_1, p) \wedge At(p, B)$ . Более того, подцель  $At(C_1, B)$  не должна быть истинной в состоянии-преемнике<sup>4</sup>. Поэтому описание состояния-преемника является таковым:

$$In(C_1, p) \wedge At(p, B) \wedge At(C_2, B) \wedge \dots \wedge At(C_{20}, B)$$

Кроме выполнения того требования, чтобы действия достигали некоторого желаемого литерала, должно также соблюдаться требование, чтобы действия не отменяли какие-либо желаемые литералы. Любое действие, удовлетворяющее этому ограничению, называется **совместимым**. Например, действие  $Load(C_2, p)$  не будет совместимым с текущей целью, поскольку оно отрицает литерал  $At(C_2, B)$ .

Составив определения понятий *релевантности* и *совместимости*, мы можем описать общий процесс формирования преемников для обратного поиска. Допустим,

---

<sup>4</sup> Если подцель была истинной в состоянии-преемнике, то данное действие все равно должно привести к целевому состоянию. С другой стороны, подобные действия являются нерелевантными, поскольку они не могут сделать цель истинной.

что при наличии описания цели  $G$  имеется действие  $A$ , которое является релевантным и совместимым. Соответствующий преемник может быть определен, как описано ниже.

- Любые положительные результаты действия  $A$ , которые появляются в цели  $G$ , удаляются.
- Добавляется каждый литерал предусловия  $A$ , если он еще не присутствует в определении действия.

Для осуществления обратного поиска могут использоваться любые из стандартных алгоритмов поиска. Завершение работы происходит после выработки такого описания преемника, которое соответствует начальному состоянию задачи планирования. В случае использования логики первого порядка для обеспечения соответствия с начальным состоянием может потребоваться подстановка переменных в описание преемника. Например, описание преемника, приведенное в предыдущем абзаце, будет соответствовать такому начальному состоянию после подстановки  $\{p/P_{12}\}$ :

$$In(C_1, P_{12}) \wedge At(P_{12}, B) \wedge At(C_2, B) \wedge \dots \wedge At(C_{20}, B)$$

Затем данная подстановка должна быть применена к действиям, ведущим из этого состояния к цели, что приводит к получению решения  $[Unload(C_1, P_{12}, B)]$ .

### Эвристики для поиска в пространстве состояний

Итак, из описанного выше следует, что ни прямой, ни обратный поиск не может быть эффективным без хорошей эвристической функции. Как было указано в главе 4, эвристическая функция оценивает расстояние от некоторого состояния до цели; в планировании Strips стоимость каждого действия равна 1, поэтому расстояние измеряется количеством действий. Основная идея состоит в том, что нужно рассматривать результаты действий и цели, которые должны быть достигнуты, а после этого выдвигать предположение, сколько действий потребуется для достижения всех целей. Задача определения точного количества действий является NP-трудной, но чаще всего существует возможность найти приемлемые оценки без слишком большого объема вычислений. Необходимо также иметь возможность вывести **допустимую** эвристику, которая не приводит к переоценке. Такая эвристика может использоваться в сочетании с поиском A\* для нахождения оптимальных решений.

Для составления эвристики могут быть опробованы два подхода. Первый из них состоит в том, чтобы вывести **ослабленную задачу** из заданной спецификации задачи (см. главу 4). Оптимальная стоимость решения для ослабленной задачи (которую можно надеяться очень легко решить) задает допустимую эвристику для первоначальной задачи. Второй подход состоит в том, что можно воспользоваться предположением о возможности применения алгоритма, действующего исключительно по принципу “разделяй и властвуй”. Такое предположение называется предположением о **независимости подцелей**, согласно которому стоимость решения конъюнкции подцелей аппроксимируется суммой стоимостей решения каждой подцели независимо друг от друга. Предположение о независимости подцелей может быть оптимистическим или пессимистическим. Оно является оптимистическим, если имеются отрицательные взаимодействия субпланов для каждой подцели, например, если действие в одном субплане удаляет цель, достигнутую в другом субплане. С другой сто-

роны, такое предположение является пессимистическим и поэтому недопустимым, если субпланы содержат избыточные действия, например, два таких действия, которые могут быть заменены одним действием в объединенном плане.

Рассмотрим способ вывода ослабленных задач планирования. Поскольку имеются явные представления предусловий и результатов, такой способ может предусматривать модификацию указанных представлений. (Сравните этот подход с задачами поиска, в которых функция определения преемника рассматривалась как “черный ящик”.) Простейшая идея состоит в том, что задача может быть ослаблена путем удаления из действий всех предусловий. В таком случае каждое действие всегда будет применимым и любой литерал может быть достигнут за один шаг (если есть какое-либо применимое действие; в противном случае цели достичь невозможно). Из этого почти безусловно следует, что количество шагов, необходимых для решения конъюнкции цели, равно количеству невыполненных целей — почти, но не совсем, поскольку, во-первых, могут существовать два таких действия, каждое из которых удаляет литерал цели, достигнутый другим, и, во-вторых, некоторое действие может достигать нескольких целей. Если же ослабленная задача будет применяться в сочетании с предположением о независимости подцелей, это позволит воспользоваться предположением о том, что указанных проблем не существует, и поэтому результирующая эвристика будет точно соответствовать количеству невыполненных целей.

Во многих случаях существует возможность получить более точную эвристику, рассматривая, по меньшей мере, положительные взаимодействия, вытекающие из наличия действий, достигающих нескольких целей. Прежде всего, можно еще больше ослабить задачу, удаляя отрицательные результаты (см. упр. 11.6). Затем можно подсчитать минимальное количество требуемых действий таким образом, чтобы объединение положительных результатов этих действий позволяло выполнить данную цель. Например, рассмотрим следующее определение:

```

Goal(A ∧ B ∧ C)
Action(X, Effect: A ∧ P)
Action(Y, Effect: B ∧ C ∧ Q)
Action(Z, Effect: B ∧ P ∧ Q)

```

Минимальное множественное покрытие цели  $\{A, B, C\}$  задается действиями  $\{X, Y\}$ , поэтому эвристика множественного покрытия возвращает стоимость 2. Такая оценка лучше по сравнению с предположением о независимости подцелей, которое позволяет получить эвристическое значение 3. Остается непреодоленной лишь одна небольшая неприятная особенность: задача поиска множественного покрытия является NP-трудной. Простой жадный алгоритм поиска множественного покрытия гарантирует возвращение значения, которое находится в пределах коэффициента  $1 + \log n$  от истинного минимального значения, где  $n$  — количество литералов в цели, но обычно на практике данный алгоритм действует намного лучше по сравнению с этой оценкой. К сожалению, жадный алгоритм не обеспечивает гарантий достижимости для такой эвристики.

Существует также возможность создавать ослабленные задачи, удаляя отрицательные результаты без удаления предусловий. Таким образом, если некоторое действие в первоначальной задаче имеет результат  $A \wedge \neg B$ , то в ослабленной задаче будет иметь результат  $A$ . Это означает, что не нужно беспокоиться об отрицательных взаимодействиях субпланов, поскольку ни одно действие не способно удалить лите-

ралы, достигнутые другим действием. Стоимость решения полученной в конечном итоге ослабленной задачи задает то, что принято называть эвристикой с ~~и~~ пустым списком удаления. Эта эвристика является весьма точной, но для ее вычисления требуется действительно выполнить некоторый (простой) алгоритм планирования. Тем не менее на практике поиск в условиях ослабленной задачи часто происходит достаточно быстро для того, чтобы имело смысл применение этой оценки стоимости.

Описанные здесь эвристики могут использоваться либо в прогрессивном, либо в регрессивном направлении. Ко времени написания данной книги первые места в рейтинге эффективности занимали прогрессивные планировщики, в которых применялась эвристика с пустым списком удаления. Такая ситуация вполне может измениться в результате исследования новых эвристик и новых методов поиска. Поскольку задачи планирования являются экспоненциально трудными<sup>5</sup>, ни один алгоритм не может быть достаточно эффективным для решения всех задач, но с использованием эвристических методов, описанных в этой главе, могут быть решены многие практические задачи, и таких задач стало гораздо больше по сравнению с теми, которые были решаемыми всего лишь несколько лет тому назад.

### 11.3. ПЛАНИРОВАНИЕ С ЧАСТИЧНЫМ УПОРЯДОЧЕНИЕМ

Прямой и обратный поиск в пространстве состояний представляют собой особые формы поиска полностью упорядоченного плана. В них рассматриваются только строго линейные последовательности действий, непосредственно связанные с начальным или целевым состоянием. Это означает, что такие методы поиска не позволяют воспользоваться преимуществами декомпозиции задачи. Вместо того чтобы обеспечить отдельную проработку каждой подзадачи, эти методы вынуждены всегда поддерживать принятие решений о том, как упорядочить действия, относящиеся ко всем подзадачам. Но обычно более предпочтительным является подход, позволяющий работать над несколькими подцелями независимо, достигать их с помощью нескольких субпланов, а затем объединять эти субпланы.

Подобный подход обладает также тем преимуществом, что позволяет добиться большей гибкости при определении последовательности, в которой составляется окончательный план. Это означает, что планировщик вначале может работать над “очевидными” или “важными” решениями, не будучи вынужденным прорабатывать все этапы в хронологическом порядке. Например, планирующий агент, который находится в Беркли и желает попасть в Монте-Карло, может вначале попытаться найти рейс из Сан-Франциско в Париж; получив информацию о времени отправления и прибытия этого рейса, он может затем заняться поиском способов того, как доехать и выехать из соответствующих аэропортов.

Общая стратегия, в которой в процессе поиска выбор определенных этапов откладывается на более позднее время, называется стратегией с ~~и~~ наименьшим вкладом (least commitment). Формального определения стратегии с наименьшим вкладом не существует, поскольку очевидно, что на любом этапе поиска должен быть сделан

<sup>5</sup> Формально процедура планирования в стиле Strips является PSPACE-полной, если только не рассматриваются действия, которые имеют только положительные предусловия и лишь один литерал результата [214].

определенный вклад в окончательное решение, так как в противном случае поиск окажется непродуктивным. Несмотря на такую неформальность, наименьший вклад — это полезная концепция для анализа того, какие решения должны быть приняты в любой задаче поиска.

Приведенный в этом разделе первый конкретный пример будет гораздо проще по сравнению с планированием отпуска, проводимого в Монте-Карло. Рассмотрим простую задачу надевания пары туфель. Ее можно описать как формальную задачу планирования следующим образом:

```
Goal(RightShoeOn ∧ LeftShoeOn)
Init()
Action(RightShoe, Precond: RightSockOn, Effect: RightShoeOn)
Action(RightSock, Effect: RightSockOn)
Action(LeftShoe, Precond: LeftSockOn, Effect: LeftShoeOn)
Action(LeftSock, Effect: LeftSockOn)
```

Планировщик должен быть способен предложить последовательность из двух действий, *RightSock* (Надеть правый носок), затем *RightShoe* (Надеть правую туфлю), чтобы достичь первого конъюнкта этой цели, и последовательность *LeftSock* (Надеть левый носок), затем *LeftShoe* (Надеть левую туфлю), чтобы достичь второго конъюнкта. После этого эти две последовательности могут быть объединены для создания окончательного плана. В ходе этого планировщик будет манипулировать двумя подпоследовательностями независимо, не задумываясь над тем, должно ли какое-то действие в одной последовательности быть выполнено до или после некоторого действия в другой. Любой алгоритм планирования, способный включить в план два действия без указания того, какое из них должно быть выполнено первым, называется **планировщиком с частичным упорядочением** (partial-order planner). На рис. 11.2 показан план с частичным упорядочением, который представляет собой решение задачи надевания туфель и носков. Обратите внимание на то, что это решение представлено в виде графа, а не последовательности действий. Заслуживают также внимания “фиктивные” действия, называемые *Start* и *Finish*, которые отмечают начало и конец плана. Назвав эти ситуации действиями, мы можем упростить структуру плана, поскольку теперь каждый этап плана становится действием. Это решение с частичным упорядочением соответствует шести возможным планам с полным упорядочением; каждый из них называется **линеаризацией** плана с частичным упорядочением.

Планирование с частичным упорядочением может быть реализовано в виде поиска в пространстве планов с частичным упорядочением. (Начиная с этого момента, мы будем называть их просто “планами”.) Это означает, что поиск начинается с пустого плана. После этого рассматриваются способы уточнения плана до тех пор, пока не удастся составить полный план, который решает данную задачу. Действия, рассматриваемые в этом поиске, являются не действиями в мире, а действиями в планах: добавление в план этапа; наложение упорядочения, согласно которому одно действие должно занять место перед другим; и т.д.

В данном разделе будет определен алгоритм POP (Partial-Order Planning) для планирования с частичным упорядочением. В соответствии с общепринятой традицией алгоритм POP оформляется в виде отдельной программы, но мы вместо этого определим задачу планирования с частичным упорядочением как экземпляр задачи поиска. Это позволит нам сосредоточиться на этапах уточнения плана, которые могут

быть применены, и не задумываться над тем, каким образом алгоритм осуществляет исследование пространства поиска. В действительности после формулировки задачи поиска может быть применен широкий перечень неинформированных или эвристических методов поиска.

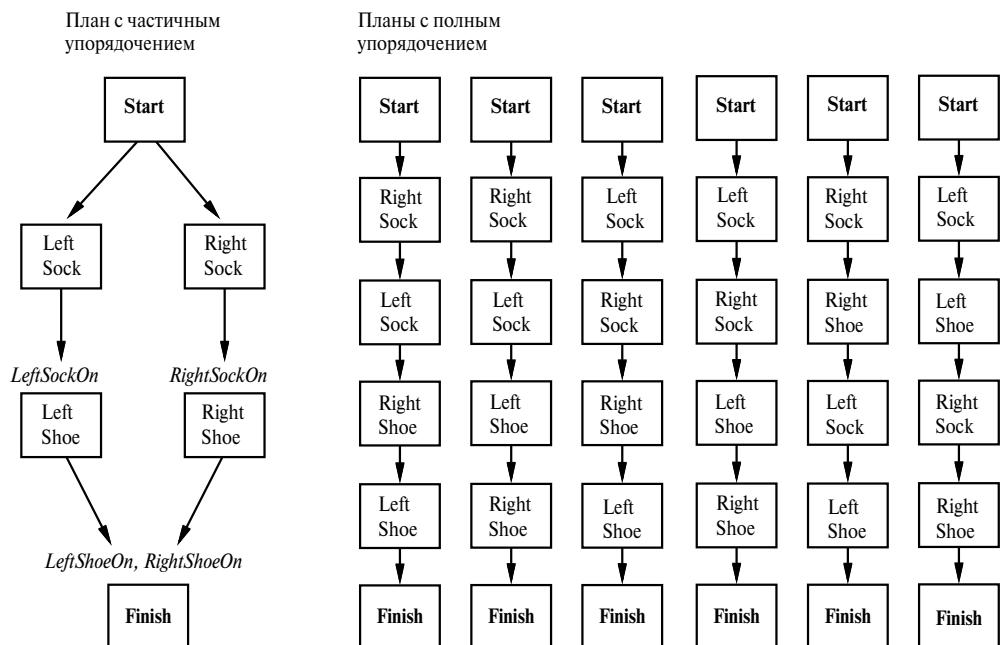


Рис. 11.2. План надевания туфель и носков с частичным упорядочением и шесть соответствующих линеаризаций в виде планов с полным упорядочением

Напомним, что состояниями рассматриваемой задачи поиска должны быть планы (в основном незаконченные). Чтобы избежать путаницы с состояниями мира, мы будем вести речь о планах, а не о состояниях. Каждый план имеет описанные ниже четыре компонента, где первые два определяют этапы плана, а последние два выполняют функции учета, позволяющие определить, как может быть дополнен план.

- Множество **действий**, из которых состоят этапы плана. Эти действия берутся из множества действий в задаче планирования. “Пустой” план содержит только действия *Start* и *Finish*. Действие *Start* не имеет предусловий, а его результатом являются все литералы в начальном состоянии задачи планирования. Действие *Finish* не имеет результатов, а его предусловиями являются литералы цели в задаче планирования.
- Множество **ограничений упорядочения**. Каждое ограничение упорядочения находится в форме  $A \prec B$ , которая читается как “*A* перед *B*” и означает, что действие *A* должно быть выполнено в какое-то время перед действием *B*, но не обязательно непосредственно перед ним. Ограничения упорядочения должны описывать правильный вариант частичного упорядочения. Любой цикл (такой как  $A \prec B$  и  $B \prec A$ ) представляет противоречие, поэтому ни одно ограничение упорядочения не может быть добавлено в план, если оно создает цикл.

- Множество  $\bowtie$  **причинных связей**. Причинная связь между двумя действиями  $A$  и  $B$  в плане записывается как  $A \xrightarrow{p} B$  и читается как “ $A \bowtie$  достигает  $p$  для  $B$ ”. Например, в следующей причинной связи:

$RightSock \xrightarrow{RightSockOn} RightShoe$

утверждается, что  $RightSockOn$  (надет правый носок) представляет собой результат действия  $RightSock$  и предусловие действия  $RightShoe$ . В ней также содержится утверждение, что предусловие  $RightSockOn$  должно оставаться истинным со времени действия  $RightSock$  до времени действия  $RightShoe$ . Другими словами, план не может быть дополнен путем добавления какого-либо нового действия  $C$ , которое  $\bowtie$  **конфликтует** с причинной связью. Действие  $C$  конфликтует со связью  $A \xrightarrow{p} B$ , если  $C$  имеет результат  $\neg p$  и если  $C$  может (в соответствии с ограничениями упорядочения) происходить после  $A$  и перед  $B$ . Некоторые авторы называют причинные связи **интервалами защиты**, поскольку связь  $A \xrightarrow{p} B$  защищает предусловие  $p$  от его отрицания в интервале от  $A$  до  $B$ .

- Множество  $\bowtie$  **открытых предусловий**. Предусловие является открытым, если оно не достигнуто с помощью какого-то действия в плане. Планировщики действуют по принципу сокращения множества открытых предусловий до пустого множества без внесения противоречия.

Например, окончательный план, показанный на рис. 11.2, имеет следующие компоненты (не показаны ограничения упорядочения, которые распространяются на каждое действие после действия  $Start$  и перед действием  $Finish$ ):

Действия:  $\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$

Упорядочения:  $\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$

Связи:  $\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,$   
 $RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$

Открытые предусловия:  $\{\}$

Мы определяем  $\bowtie$  **согласованный план** (consistent plan) как план, в котором не имеется циклов в ограничениях упорядочения и нет конфликтов с причинными связями. Согласованный план без открытых предусловий представляет собой **решение**. Даже краткие размышления должны убедить читателя в справедливости следующего утверждения:  $\bowtie$  *каждая линеаризация решения с частичным упорядочением представляет собой решение с полным упорядочением, выполнение которого из начального состояния позволяет достичь целевого состояния*. Это означает, что можно распространить понятие “выполнения плана” с планов с полным упорядочением на планы с частичным упорядочением. План с частичным упорядочением выполняется путем повторного выбора любого из возможных следующих действий. Как будет показано в главе 12, такая гибкость, предоставляемая агенту в ходе выполнения плана, может быть очень полезной в тех обстоятельствах, когда мир к нему не благосклонен. Гибкое упорядочение позволяет также упростить комбинирование меньших планов в более крупные, поскольку каждый из меньших планов допускает переупорядочение его действий для предотвращения конфликтов с другими планами.

Теперь мы готовы сформулировать задачу поиска, которую решает алгоритм POP, как показано ниже. Начнем с формулировки, подходящей для задач планирования в пропозициональной логике, оставляя на будущее осложнения, связанные с использо-

зованием логики первого порядка. Как обычно, это определение включает начальное состояние, действия и проверку цели.

- Начальный план содержит действия *Start* и *Finish*, ограничение упорядочения  $Start \prec Finish$ , не включает ни одной причинной связи и имеет все предусловия в действии *Finish* в качестве открытых предусловий.
- Функция определения преемника произвольным образом выбирает одно открытое предусловие  $p$  действия *B* и вырабатывает план-преемник для каждого возможного согласованного способа выбора действия *A*, которое достигает  $p$ . Согласованность обеспечивается принудительно следующим образом.
  1. Причинная связь  $A \xrightarrow{p} B$  и ограничение упорядочения  $A \prec B$  добавляются к плану. Действие *A* может быть существующим действием в плане или новым действием. Если оно является новым, добавить его к плану, а также добавить  $Start \prec A$  и  $A \prec Finish$ .
  2. Разрешаются конфликты между новой причинной связью и всеми существующими действиями, а также между действием *A* (если оно является новым) и всеми существующими причинными связями. Конфликт между  $A \xrightarrow{p} B$  и *C* разрешается путем обеспечения того, чтобы действие *C* происходило в какое-то время за пределами интервала защиты, либо за счет добавления ограничения  $B \prec C$ ; либо путем добавления ограничения  $C \prec A$ . Для одного из них или обоих добавляются состояния-преемники, если они приводят к созданию согласованных планов.
- В ходе проверки цели осуществляется проверка того, является ли текущий план решением первоначальной задачи планирования. Поскольку вырабатываются только согласованные планы, в проверке цели достаточно только проанализировать, есть ли еще открытые предусловия.

Еще раз напомним, что действия, рассматриваемые в алгоритмах поиска при такой формулировке, представляют собой этапы уточнения плана, а не реальные действия из самой проблемной области. Поэтому, строго говоря, стоимость пути не имеет значения, поскольку важна только суммарная стоимость реальных действий в плане, к которому ведет этот путь. Тем не менее существует возможность определить функцию стоимости пути, которая отражает стоимости реальных планов; для этого можно назначать цену 1 за каждое реальное действие, добавленное к плану, и 0 — за все остальные этапы уточнения. В таком случае значение функции  $g(n)$ , где  $n$  — некоторый план, будет равно количеству реальных действий в плане. Может также использоваться эвристическая оценка  $h(n)$ .

На первый взгляд может показаться, что функция определения преемника должна вырабатывать преемников для каждого открытого предусловия  $p$ , а не только для одного из них. Но такой подход был бы избыточным и неэффективным по той же причине, по которой алгоритмы удовлетворения ограничений не включают преемников для каждой возможной переменной, поскольку порядок, в котором рассматриваются открытые предусловия (как и порядок, в котором рассматриваются переменные CSP), является коммутативным (см. с. 214). Таким образом, мы можем выбирать произвольное упорядочение и все еще иметь полный алгоритм. Выбор правильного упорядочения может привести к более быстрому поиску, но все упорядочения оканчиваются одним и тем же множеством решений-кандидатов.

### Пример планирования с частичным упорядочением

Теперь рассмотрим, как можно с помощью алгоритма POP решить задачу с запасным колесом, описанную в разделе 11.1. Описание этой задачи повторно приведено в листинге 11.4.

#### Листинг 11.4. Описание простой задачи замены колеса со стертой покрышкой

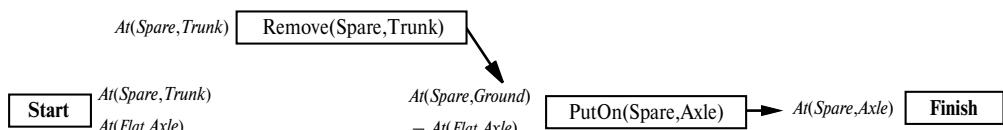
```

Init(At(Flat,Axle) ∧ At(Spare,Trunk))
Goal(At(Spare,Axle))
Action(Remove(Spare,Trunk),
    Precond: At(Spare,Trunk)
    Effect: ¬At(Spare,Trunk) ∧ At(Spare,Ground))
Action(Remove(Flat,Axle),
    Precond: At(Flat,Axle)
    Effect: ¬At(Flat,Axle) ∧ At(Flat,Ground))
Action(PutOn(Spare, Axle),
    Precond: At(Spare,Ground) ∧ ¬At(Flat,Axle)
    Effect: ¬At(Spare,Ground) ∧ At(Spare,Axle))
Action(LeaveOvernight,
    Precond:
    Effect: ¬At(Spare,Ground) ∧ ¬At(Spare,Axle) ∧ ¬At(Spare,Trunk)
          ∧ ¬At(Flat,Ground) ∧ ¬At(Flat,Axle))

```

Поиск решения начинается с начального плана, содержащего действие *Start* с результатом *At(Spare, Trunk)* ∧ *At(Flat, Axle)* и действие *Finish* с единственным предусловием *At(Spare, Axle)*. Затем вырабатываются преемники путем взятия открытого предусловия для его проработки (не допускающей отмены) и выбора среди возможных действий для его достижения. На данный момент мы не будем задумываться над тем, что нужно воспользоваться какой-то эвристической функцией, которая могла бы помочь в выработке этих решений, и станем выбирать варианты, которые внешне кажутся произвольными. Последовательность событий описана ниже.

1. Взять единственное открытое предусловие, *At(Spare, Axle)*, действия *Finish*. Выбрать единственное применимое действие, *PutOn(Spare, Axle)*.
2. Взять предусловие *At(Spare, Ground)* действия *PutOn(Spare, Axle)*. Выбрать единственное применимое действие, *Remove(Spare, Trunk)*, чтобы достичь его. Результатирующий план показан на рис. 11.3.



*Рис. 11.3. Незаконченный план с частичным упорядочением для задачи замены колеса после выбора действий, соответствующих первым двум открытым предусловиям. В прямоугольниках показаны действия, предусловия которых находятся слева, а результаты — справа (результаты не показаны, кроме результатов действия *Start*). Сплошные линии со стрелками показывают причинные связи, защищающие высказывание, находящееся у острия стрелки*

3. Взять предусловие  $\neg At(Flat, Axle)$  действия *PutOn(Spare, Axle)*. Просто для того чтобы поступить вопреки здравому смыслу, выберем действие *LeaveOvernight*, а не действие *Remove(Flat, Axle)*. Но обратите внимание на то, что действие *LeaveOvernight* также имеет результат  $\neg At(Spare, Ground)$ , который означает, что оно конфликтует со следующей причинной связью:

*Remove(Spare, Trunk)  $\xrightarrow{At(Spare, Ground)}$  PutOn(Spare, Axle)*

Чтобы разрешить этот конфликт, добавим ограничение упорядочения, которое помещает действие *LeaveOvernight* перед действием *Remove(Spare, Trunk)*. Возникающий при этом план показан на рис. 11.4. (Предлагаем читателю ответить на вопросы, почему такое дополнение позволяет разрешить конфликт и почему нет другого способа его разрешить?)

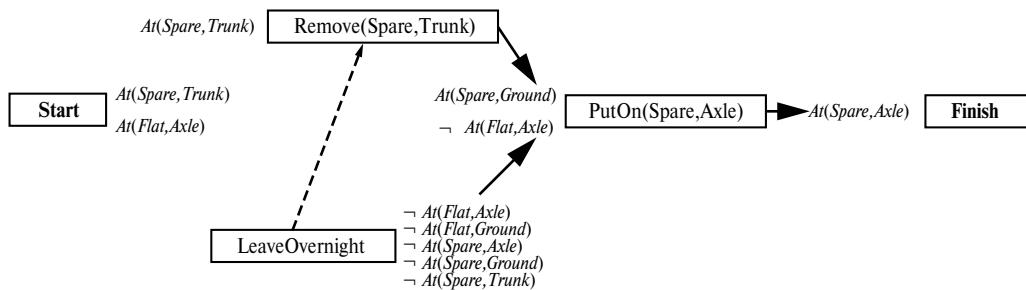


Рис. 11.4. План, возникающий после выбора *LeaveOvernight* в качестве действия для достижения предусловия  $\neg At(Flat, Axle)$ . Чтобы избежать конфликта с причинной связью, исходящей из действия *Remove(Spare, Trunk)*, которая защищает действие *At(Spare, Ground)*, на *LeaveOvernight* накладывается ограничение, чтобы это действие происходило перед *Remove(Spare, Trunk)*, как показано штриховой линией со стрелкой

4. В этот момент единственным оставшимся открытым предусловием является предусловие *At(Spare, Trunk)* действия *Remove(Spare, Trunk)*. Единственным действием, позволяющим достичь этого предусловия, является существующее действие *Start*, но причинная связь от *Start* к *Remove(Spare, Trunk)* конфликтует с результатом  $\neg At(Spare, Trunk)$  действия *LeaveOvernight*. В данный момент не существует способа разрешить конфликт с действием *LeaveOvernight* — его нельзя переупорядочить таким образом, чтобы оно находилось перед *Start* (поскольку ни одно действие не может происходить перед действием *Start*), и нельзя переупорядочить его так, чтобы оно находилось после *Remove(Spare, Trunk)* (поскольку уже имеется ограничение, которое упорядочивает его перед *Remove(Spare, Trunk)*). Поэтому необходимо вернуться к одному из предыдущих состояний, удалить действие *LeaveOvernight* и две последние причинные связи, возвратившись в состояние, показанное на рис. 11.3. Планировщик фактически доказал, что действие *LeaveOvernight* не может применяться в качестве способа замены колеса.

5. Еще раз рассмотрим предусловие  $\neg At(Flat, Axle)$  действия  $PutOn(Spare, Axle)$ . На этот раз выберем действие  $Remove(Flat, Axle)$ .
6. Снова возьмем предусловие  $At(Spare, Trunk)$  действия  $Remove(Spare, Trunk)$  и выберем действие  $Start$ , чтобы его достичь. На сей раз конфликты не возникают.
7. Возьмем предусловие  $At(Flat, Axle)$  действия  $Remove(Flat, Axle)$  и выберем действие  $Start$  для его достижения. Это дает нам полный согласованный план (иными словами, решение), как показано на рис. 11.5.

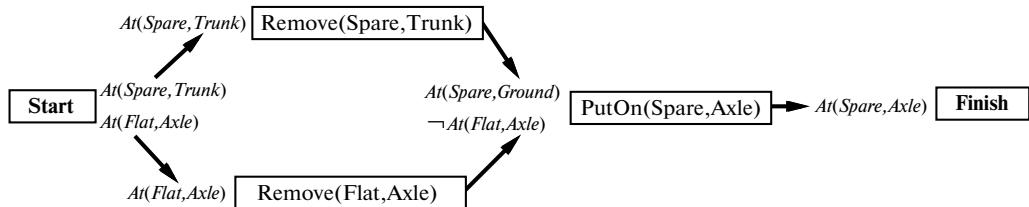


Рис. 11.5. Окончательное решение задачи замены колеса. Обратите внимание на то, что действия  $Remove(Spare, Trunk)$  и  $Remove(Flat, Axle)$  могут выполняться в любом порядке, при условии, что они завершаются перед действием  $PutOn(Spare, Axle)$

Хотя данный пример очень прост, он иллюстрирует некоторые преимущества планирования с частичным упорядочением. Во-первых, причинные связи способствуют заблаговременному отсечению тех частей пространства поиска, которые не содержат решений из-за неустранимых конфликтов. Во-вторых, решение, приведенное на рис. 11.5, представляет собой план с частичным упорядочением. В данном случае преимущества невелики, поскольку имеются только две возможные линеаризации; тем не менее повышенная гибкость выбора вариантов может пойти агенту на пользу, например, если ему придется менять колесо в середине шоссе с интенсивным движением.

Этот пример также указывает на некоторые возможные усовершенствования, которые вполне могут быть осуществлены. Например, в данном случае налицо дублирование усилий: действие  $Start$  соединяется связью с действием  $Remove(Spare, Trunk)$  до того, как конфликт вызывает возврат, а затем эта связь снова разрывается из-за возврата, даже несмотря на то, что она не участвует в конфликте. После этого в ходе продолжения поиска связь снова восстанавливается. Такое развитие событий является типичным для хронологических возвратов, и его последствия могут быть смягчены с помощью возвратов, управляемых с учетом зависимостей.

### Планирование с частичным упорядочением и несвязанными переменными

В этом разделе рассматриваются сложности, которые могут возникнуть, если алгоритм POP используется с представлениями действий в логике первого порядка, которые могут включать переменные. Предположим, что имеется задача в мире блоков (см. листинг 11.3) с открытым предусловием  $On(A, B)$  и следующим действием:

```

Action(Move(b, x, y),
Precond: On(b, x) ∧ Clear(b) ∧ Clear(y),
Effect: On(b, y) ∧ Clear(x) ∧ ¬On(b, x) ∧ ¬Clear(y))
  
```

Это действие достигает результата  $On(A, B)$ , поскольку результат  $On(b, y)$  унифицируется с термом  $On(A, B)$  с помощью подстановки  $\{b/A, y/B\}$ . Затем применим эту подстановку к действию и получим следующее:

```
Action Move(A, x, B)
Precond: On(A, x) ∧ Clear(A) ∧ Clear(B)
Effect: On(A, B) ∧ Clear(x) ∧ ¬On(A, x) ∧ ¬Clear(B))
```

При этом переменная  $x$  остается несвязанной. Тем самым в данном действии определено, что блок  $A$  нужно переместить откуда-то, а откуда, не сказано. В этом заключается еще один пример реализации принципа наименьшего вклада: мы можем отложить выполнение выбора до какого-то этапа, в котором сам план укажет для нас этот выбор. Например, предположим, что в начальном состоянии имеется предусловие  $On(A, D)$ . В таком случае для достижения предусловия  $On(A, x)$  может использоваться действие *Start* со связыванием  $x$  с  $D$ . Такая стратегия с ожиданием дополнительной информации перед выбором  $x$  часто является более эффективной по сравнению с опробованием любого возможного значения  $x$  и возвратом в ответ на неудачный выбор каждого из этих значений.

Из-за наличия переменных в предусловиях и действиях процесс обнаружения и разрешения конфликтов усложняется. Например, после добавления к плану действия  $Move(A, x, B)$  потребуется следующая причинная связь:

$$Move(A, x, B) \xrightarrow{On(A, B)} Finish$$

Если имеется другое действие  $M_2$  с результатом  $\neg On(A, z)$ , то  $M_2$  конфликтует, только если переменная  $z$  равна  $B$ . Чтобы учесть такую возможность, расширим представление планов для включения множества  $\text{ограничений неравенства}$  в форме  $z \neq X$ , где  $z$  — переменная, а  $X$  — либо другая переменная, либо константный символ. В данном случае можно было бы разрешить конфликт путем добавления ограничения  $z \neq B$ , которое означает, что в будущих дополнениях к плану разрешается конкретизировать переменную  $z$  любым значением, кроме  $B$ . При каждом применении к плану некоторой подстановки необходимо контролировать, чтобы подобные неравенства не противоречили этой подстановке. Например, подстановка, которая включает  $x/y$ , конфликтует с ограничением неравенства  $x \neq y$ . Такие конфликты не могут быть разрешены, поэтому планировщику приходится выполнять возврат.

Более подробный пример планирования по алгоритму РОР с переменными в мире блоков приведен в разделе 12.6.

## Эвристики для планирования с частичным упорядочением

В отличие от планирования с полным упорядочением, планирование с частичным упорядочением обладает явным преимуществом, поскольку позволяет выполнять декомпозицию задач на подзадачи. Оно имеет также определенный недостаток, который заключается в том, что состояния не представлены явно, поэтому труднее оценить, насколько далек план с частичным упорядочением от ситуации достижения цели. К тому же в настоящем времени существует гораздо меньшее понимание того, как следует вычислять точные эвристики для планирования с частичным упорядочением, чем для планирования с полным упорядочением.

Наиболее очевидная эвристика состоит в подсчете количества открытых предусловий. Такая эвристика может быть улучшена путем вычитания из указанной величины количества открытых предусловий, которые согласуются с литералами в состоянии *Start*. Как и в случае с полным упорядочением, такая эвристика приводит к переоценке стоимости, если имеются действия, достигающие нескольких подцелей, и недооценке стоимости, если возникают отрицательные взаимодействия этапов плана. В следующем разделе представлен подход, позволяющий получать гораздо более точные эвристики из ослабленной задачи.

Эвристическая функция используется для выбора плана, подлежащего уточнению. При наличии такого выбора алгоритм вырабатывает преемников на основе определения единственного открытого предусловия, которое следует дополнительно проработать. Как и в случае выбора переменной в алгоритмах удовлетворения ограничений, такое решение оказывает большое влияние на эффективность. Для алгоритмов планирования может быть адаптирована эвристика с **наиболее ограниченной переменной**, применяемая в задачах CSP, и эта эвристика, по-видимому, неплохо действует. Ее идея состоит в том, чтобы определить открытое условие, которое может быть выполнено с помощью наименьшего количества способов. Возникают два особых случая применения этой эвристики. Во-первых, если какого-то открытого условия невозможно достичь с помощью любого действия, оно должно быть выбрано данной эвристикой; это — вполне целесообразно, поскольку раннее обнаружение неразрешимой ситуации позволяет сэкономить большой объем работы. Во-вторых, если открытое условие может быть достигнуто только одним способом, оно также должно быть выбрано, поскольку такое решение неизбежно и позволяет налагать дополнительные ограничения на другие варианты выбора, которые еще должны быть сделаны. Хотя процедура полного вычисления количества способов выполнения каждого открытого условия является дорогостоящей и не всегда оправданной, эксперименты показали, что применение этих двух частных случаев обеспечивает весьма существенное ускорение.

## 11.4. ГРАФЫ ПЛАНИРОВАНИЯ

Все предложенные выше эвристики для планирования с полным упорядочением и частичным упорядочением могут допускать неточности. В данном разделе показано, как можно воспользоваться специальной структурой данных, называемой **графом планирования**, для получения лучших эвристических оценок. Такие эвристики могут применяться в любом из методов поиска, рассмотренных нами до сих пор. Еще один вариант состоит в том, что решение может быть извлечено непосредственно из графа планирования с использованием специализированного алгоритма, например такого, который получил название *Graphplan*.

Граф планирования состоит из последовательности **уровней**, которые соответствуют временным этапам в плане, где уровень 0 представляет собой начальное состояние. Каждый уровень содержит множество литералов и множество действий. Грубо говоря, литералы представляют собой то, что может быть истинным на каждом временном этапе в зависимости от действий, выполненных на предыдущих временных этапах. Также грубо говоря, действиями являются все те действия, которые

могут иметь все свои предусловия выполнеными на данном временном этапе в зависимости от того, какие из литералов фактически являются истинными. В этих двух определениях применяется выражение “грубо говоря”, поскольку в графе планирования регистрируется только ограниченное подмножество возможных отрицательных взаимодействий между действиями; поэтому граф может давать оптимистическую оценку минимального количества временных этапов, требуемых для того, чтобы некоторый литерал стал истинным. Тем не менее такое количество этапов в графе планирования предоставляет хорошую оценку того, насколько трудно достичь указанного литерала из начального состояния. Еще важнее то, что граф планирования определен таким образом, что может быть сформирован очень эффективно.

Графы планирования могут применяться только для решения задач планирования в пропозициональной логике, т.е. тех задач, в которых отсутствуют переменные. Как упоминалось в разделе 11.1, в пропозициональную форму могут быть преобразованы и представления Strips, и представления ADL. Если в задаче имеется большое количество объектов, это может привести к весьма существенному возрастанию количества возможных схем действий. Несмотря на это, было показано, что графы планирования представляют собой эффективное инструментальное средство решения трудных задач планирования.

Проиллюстрируем применение графов планирования на простом примере. (Использование более сложных примеров привело бы к созданию графов, которые не поместились бы на одной странице книги.) В листинге 11.5 приведена задача, а на рис. 11.6 показан ее граф планирования. Начнем с уровня состояния  $S_0$ , который представляет начальное состояние задачи. Затем перейдем на уровень действия  $A_0$  и поместим на него все действия, предусловия которых были выполнены на предыдущем уровне. Каждое действие соединено с его предусловиями в состоянии  $S_0$  и его результатами в состоянии  $S_1$ , а это в данном случае влечет за собой введение в  $S_1$  новых литералов, которых не было в  $S_0$ .

#### Листинг 11.5. Задача “получить кекс, а также съесть кекс”

---

```

Init(Have(Cake))
Goal(Have(Cake) ∧ Eaten(Cake))

Action(Eat(Cake)
      Precond: Have(Cake)
      Effect: ¬Have(Cake) ∧ Eaten(Cake))

Action(Bake(Cake)
      Precond: ¬Have(Cake)
      Effect: Have(Cake))

```

---

Должны быть предусмотрены способы представлять на графике планирования не только действие, но и бездействие. Это означает, что необходимо иметь своего рода эквивалент аксиом окружения ситуационного исчисления, которые позволяют литералу оставаться истинным от одной ситуации к другой, если его не изменяет ни одно действие. В графике планирования это осуществляется с помощью множества **сохраняющих действий** (persistence action). Для каждого положительного или отрицательного литерала  $C$  в задачу вводится сохраняющее действие с предусловием  $C$  и результатом  $C$ . На рис. 11.6 в состоянии  $A_0$  показано одно “реальное” действие,

*Eat(Cake)* (съесть кекс), наряду с двумя сохраняющими действиями, обозначенными с помощью небольших квадратов.

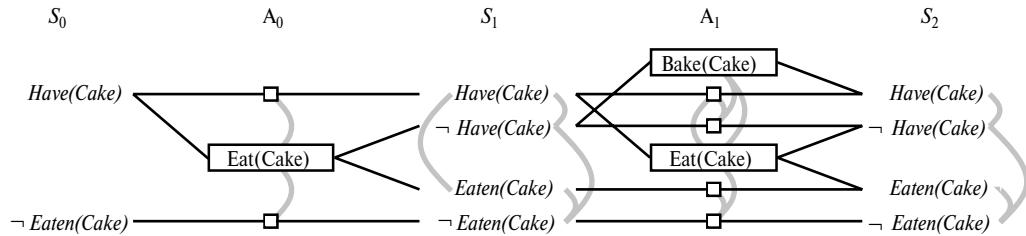


Рис. 11.6. Граф планирования для задачи “получить кекс, а также съесть кекс” вплоть до уровня  $S_2$ . Прямоугольники обозначают действия (маленькие квадраты обозначают сохраняющие действия), а прямые линии указывают на предусловия и результаты. Взаимно исключающие связи обозначены кривыми серыми линиями

Уровень  $A_0$  содержит все действия, которые могут произойти в состоянии  $S_0$ , но столь же важно то, что он показывает конфликты между действиями, способные воспрепятствовать тому, чтобы эти действия выполнялись вместе. Серыми линиями на рис. 11.6 показаны **взаимно исключающие связи** (или связи, **характеризующиеся взаимным исключением** — mutual exclusive, mutex). Например, действие *Eat(Cake)* является взаимно исключающим с действием, обеспечивающим сохранение состояния, вызванного действием *Have(Cake)* или  $\neg Eaten(Cake)$ . Вскоре будет показано, как происходит вычисление взаимно исключающих связей.

На уровне  $S_1$  находятся все литералы, которые могут стать результатом принятия к исполнению любого подмножества действий уровня  $A_0$ . На этом уровне показаны также взаимно исключающие связи (серые линии), обозначающие литералы, которые не могут появляться вместе, независимо от выбора действий. Например, взаимно исключающими являются литералы *Have(Cake)* и *Eaten(Cake)*: в зависимости от выбора действий на уровне  $A_0$  результатом может стать один из них или другой, но не оба. Иными словами, на уровне  $S_1$  представлено несколько состояний, точно так же, как и при регрессивном поиске в пространстве состояний, а взаимно исключающие связи представляют собой ограничения, которые определяют множество возможных состояний.

Построение плана продолжается таким же образом, с чередованием между уровнями состояния  $S_i$  и уровнями действия  $A_i$ , до тех пор, пока не будет достигнут уровень, на котором два последовательных уровня становятся идентичными. После достижения такой точки граф называется **выровненным** (leveled off). Каждый последующий уровень будет оставаться неизменным, поэтому дальнейшее развертывание не имеет смысла.

В конечном итоге образуется структура, в которой каждый уровень  $A_i$  содержит все действия, применимые на уровне  $S_i$ , наряду с ограничениями, указывающими, какие пары действий не могут выполняться одновременно. Каждый уровень  $S_i$  содержит все литералы, которые могут стать результатом любого возможного выбора действий на уровне  $A_{i-1}$ , наряду с ограничениями, указывающими, какие пары литералов недопустимы. Важно отметить, что процесс построения графа планирования не требует выбора среди действий, который потребовал бы комбинаторного поиска. Вместо этого в

графе просто регистрируется невозможность осуществления некоторых выборов с использованием взаимно исключающих связей. Сложность формирования графа планирования оценивается полиномиальной зависимостью низкого порядка от количества действий и литералов, тогда как размеры пространства состояний определяются экспоненциальной зависимостью от количества литералов.

Теперь определим взаимно исключающие связи как для действий, так и для литералов. Между двумя действиями на данном конкретном уровне имеет место взаимно исключающее отношение, если соблюдается любое из трех перечисленных ниже условий.

- **Несогласованные результаты.** Одно действие отрицает результат другого. Например, действие *Eat(Cake)* и сохраняющее действие *Have(Cake)* имеют несогласованные результаты, поскольку они не согласуются в результате *Have(Cake)* (съедение кекса приводит к его исчезновению).
- **Вмешательство.** Один из результатов одного действия является отрицанием предусловия другого действия. Например, действие *Eat(Cake)* мешает осуществлению сохраняющего действия *Have(Cake)*, поскольку отрицает его предусловие (если кекса больше нет, то нечего и хранить).
- **Конкурирующие потребности.** Одно из предусловий одного действия является взаимно исключающим по отношению к предусловию другого. Например, литералы *Bake(Cake)* (Испечь кекс) и *Eat(Cake)* (Съесть кекс) являются взаимно исключающими, поскольку конкурируют за значение предусловия *Have(Cake)* (в одном литерале кекс создается, а в другом — уничтожается).

Взаимно исключающее отношение имеет место между двумя литералами на одном и том же уровне, если один из них является отрицанием другого или если каждая возможная пара действий, которые позволяют достичь двух литералов, является взаимно исключающей. Такое условие называется *несогласованной поддержкой*. Например, на уровне  $S_1$  литералы *Have(Cake)* и *Eaten(Cake)* являются взаимно исключающими, поскольку единственный способ достичь литерала *Have(Cake)* (применения сохраняющего действия) является взаимно исключающим с единственным способом достижения литерала *Eaten(Cake)*, а именно *Eat(Cake)*. На уровне  $S_2$  эти два литерала не являются взаимно исключающими, поскольку существуют новые способы их достижения, такие как действие *Bake(Cake)* и сохраняющее действие *Eaten(Cake)*, которые не являются взаимно исключающими.

### Применение графов планирования для получения эвристической оценки

Граф планирования после его построения становится богатым источником информации о задаче. Например, очевидно, что ~~литерал, который не появляется на заключительном уровне графа, не может быть достигнут с помощью любого плана~~. Такое наблюдение может использоваться в обратном поиске следующим образом: любое состояние, содержащее недостижимый литерал, имеет стоимость  $h(n) = \infty$ . Аналогичным образом, при планировании с частичным упорядочением любой план с недостижимым открытым условием имеет  $h(n) = \infty$ .

Эту идею можно сделать более обобщенной. Стоимость достижения любого целевого литерала может оцениваться как номер уровня, на котором он впервые появ-

ляется в графе планирования. Назовем эту оценку **уровневой стоимостью** (level cost) цели. На рис. 11.6 литерал *Have(Cake)* имеет уровневую стоимость 0, а *Eaten(Cake)* — уровневую стоимость 1. Можно легко показать (упр. 11.9), что эти оценки являются допустимыми для отдельных целей. Однако сами эти оценки могут оказаться не очень качественными, поскольку графы планирования допускают наличие нескольких действий на каждом уровне, тогда как в этой эвристике учитывается только номер уровня, а не количество действий. По этой причине для вычисления эвристики принято использовать **последовательный график планирования** (serial planning graph). Последовательный график требует, чтобы на каждом конкретном временном этапе фактически могло происходить только одно действие; такое требование осуществляется путем введения взаимно исключающих связей между каждой парой действий, кроме сохраняющих действий. Уровневые стоимости, извлеченные из последовательных графов, часто представляют собой вполне приемлемые оценки фактических стоимостей.

Чтобы оценивать стоимость достижения конъюнкции целей, можно воспользоваться одним из трех простых подходов. В эвристике **максимального уровня** (max-level) просто берется максимальная уровневая стоимость любой из целей; такая эвристика является допустимой, но не обязательно очень точной. Эвристика **уровневой суммы** (level sum), в основе которой лежит предположение о независимости подцелей, возвращает сумму уровневых стоимостей целей; эта эвристика является недопустимой, но очень хорошо действует на практике при решении задач, которые являются в значительной степени декомпонуемыми. Она характеризуется гораздо более высокой точностью по сравнению с эвристикой, в которой учитывается количество невыполненных целей, описанной в разделе 11.2. В рассматриваемой задаче эвристическая оценка для конъюнктивной цели *Have(Cake)  $\wedge$  Eaten(Cake)* будет равна  $0+1=1$ , тогда как правильный ответ равен 2. Кроме того, если будет удалено действие *Bake(Cake)*, эта оценка по-прежнему будет равна 1, но достижение этой конъюнктивной цели станет невозможной. Наконец, эвристика **множественного уровня** (set-level) находит уровень, на котором все литералы в конъюнктивной цели появляются в графике планирования без любой пары из них, которая была бы взаимно исключающей. Эта эвристика дает правильное значение, равное 2, для первоначальной задачи и равное бесконечности для задачи без действия *Bake(Cake)*. Она доминирует над эвристикой максимального уровня и действует чрезвычайно успешно в задачах, характеризующихся весьма существенным взаимодействием субпланов.

Для использования его в качестве инструментального средства формирования точных эвристик график планирования можно рассматривать как ослабленную задачу, которая может быть эффективно решена. Чтобы понять характер этой ослабленной задачи, нужно точно определить, что означает появление литерала  $g$  на уровне  $S_i$  в графике планирования. В идеале желательно было бы иметь гарантию, что существует план с  $i$  уровнями действий, который достигает литерала  $g$ , а также, что литерал  $g$  не появится, если такого плана нет. К сожалению, предоставить такую гарантию столь же трудно, как и решить первоначальную задачу планирования. Поэтому график планирования предоставляет вторую половину гарантии (если литерал  $g$  не появляется, то нет и плана его достижения), но если литерал  $g$  появляется, то весь этот график планирования становится залогом того, что существует план, который, возможно, позволяет достичь литерала  $g$  и не имеет “очевидных” недостатков. Очевид-

**ный недостаток** плана определяется как недостаток, который может быть выявлен путем рассмотрения двух действий или двух литералов одновременно; другими словами, путем проверки взаимно исключающих отношений. Могут существовать более трудно диагностируемые недостатки, охватывающие три, четыре или больше действий, но опыт показывает, что вычислительные затраты, связанные с отслеживанием этих возможных недостатков, не оправдываются. Этот вывод аналогичен уроку, усвоенному по результатам исследования задач удовлетворения ограничений, в которых часто целесообразно вычислить 2-совместимость (совместимость на уровне 2) перед поиском решения, но вычисление 3-совместимости или совместимости более высокой степени часто бывает менее целесообразным (см. раздел 5.2).

### Алгоритм Graphplan

В данном подразделе показано, как извлечь план непосредственно из графа планирования, а не просто использовать этот график для получения эвристики. Алгоритм Graphplan (листинг 11.6) имеет два основных этапа, каждый из которых чередуется в цикле. Прежде всего в этом алгоритме проверяется, присутствуют ли все целевые литералы на текущем уровне без взаимно исключающих связей между любой парой из них. Если это требование соблюдается, то в текущем графике может существовать решение, поэтому в алгоритме выполняется попытка извлечь это решение. В противном случае график расширяется путем добавления действий для текущего уровня и литералов состояния для следующего уровня. Процесс продолжается до тех пор, пока либо не обнаруживается решение, либо не выясняется, что решения не существует.

**Листинг 11.6. Алгоритм Graphplan.** В алгоритме Graphplan чередуются этап извлечения решения и этап расширения графа. Функция Extract-Solution определяет, может ли быть найден план, начиная с конца и выполняя поиск в обратном направлении. Функция Expand-Graph добавляет действия для текущего уровня и литералы состояния для следующего уровня

---

```

function Graphplan(problem) returns решение solution или индикатор
неудачи failure

graph  $\leftarrow$  Initial-Planning-Graph(problem)
goals  $\leftarrow$  Goals[problem]
loop do
    if все цели goals не являются взаимно исключающими
        на последнем уровне графа graph then do
            solution  $\leftarrow$  Extract-Solution(graph, goals, Length(graph))
            if solution  $\neq$  failure then return solution
            else if No-Solution-Possible(graph) then return failure
    graph  $\leftarrow$  Expand-Graph(graph, problem)

```

---

Теперь проследим за функционированием алгоритма Graphplan на примере задачи замены колеса, рассматриваемой в разделе 11.1. Полный график показан на рис. 11.7. В первой строке алгоритма Graphplan график планирования инициализируется значением одноуровневого графа ( $S_0$ ), состоящего из пяти литералов, взятых из начального состояния. Целевой литерал *At(Spare, Axle)* в состоянии  $S_0$  отсутствует, поэтому не требуется вызывать функцию Extract-Solution — мы можем быть уверены, что решение еще не существует. Вместо этого вызывается функция

Expand-Graph, добавляющая три действия, предусловия которых существуют на уровне  $S_0$  (т.е. все действия, за исключением  $\text{PutOn}(\text{Spare}, \text{Axle})$ ), наряду с сохраняющими действиями для всех литералов в  $S_0$ . Результаты действий добавляются на уровне  $S_1$ . Затем функция Expand-Graph отыскивает взаимно исключающие отношения и добавляет их к графу.

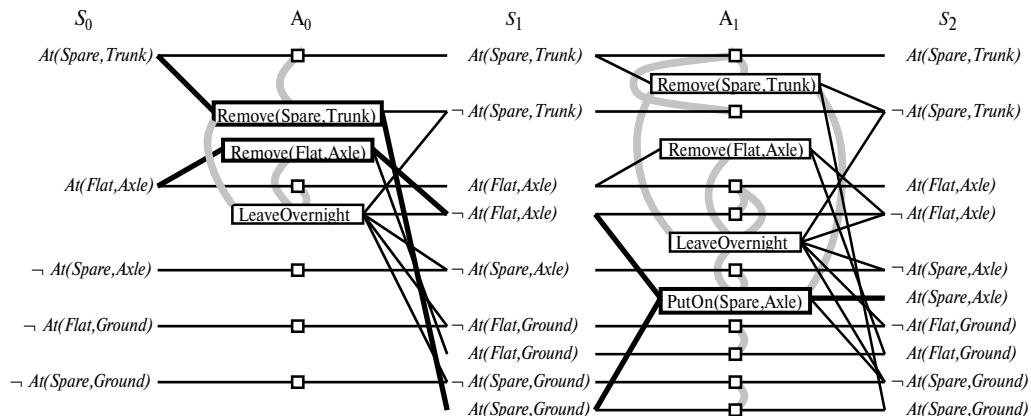


Рис. 11.7. Граф планирования для задачи замены колеса после расширения до уровня  $S_2$ . Взаимно исключающие связи показаны серыми линиями. Показаны только некоторые наиболее важные взаимно исключающие связи, поскольку граф был бы слишком загроможден, если бы были показаны все эти связи. Решение обозначено жирными линиями и жирными контурами прямоугольников

В состоянии  $S_1$  литерал  $\text{At}(\text{Spare}, \text{Axle})$  все еще не присутствует, поэтому функция Extract-Solution снова не вызывается. Вызов функции Expand-Graph приводит к получению графа планирования, показанного на рис. 11.7. Теперь, после получения полного комплекта действий, целесообразно рассмотреть некоторые примеры взаимно исключающих отношений и их причин, как показано ниже.

- Несогласованные результаты.** Литерал  $\text{Remove}(\text{Spare}, \text{Trunk})$  является взаимно исключающим по отношению к  $\text{LeaveOvernight}$ , поскольку один из них имеет своим результатом литерал  $\text{At}(\text{Spare}, \text{Ground})$ , а другой — его отрицание.
- Вмешательство.** Литерал  $\text{Remove}(\text{Flat}, \text{Axle})$  является взаимно исключающим по отношению к  $\text{LeaveOvernight}$ , поскольку один из них имеет своим предусловием  $\text{At}(\text{Flat}, \text{Axle})$ , а другой имеет своим результатом его отрицание.
- Конкурирующие потребности.** Литерал  $\text{PutOn}(\text{Spare}, \text{Axle})$  является взаимно исключающим по отношению к  $\text{Remove}(\text{Flat}, \text{Axle})$ , поскольку один из них имеет в качестве предусловия литерал  $\text{At}(\text{Flat}, \text{Axle})$ , а другой — его отрицание.
- Несогласованная поддержка.** Литерал  $\text{At}(\text{Spare}, \text{Axle})$  является взаимно исключающим по отношению к  $\text{At}(\text{Flat}, \text{Axle})$  на уровне  $S_2$ , поскольку единственным способом достижения цели  $\text{At}(\text{Spare}, \text{Axle})$  является выполнение действия  $\text{PutOn}(\text{Spare}, \text{Axle})$ , а оно является взаимно исключающим с сохраняющим действием, которое представляет собой единственный способ достижения литерала  $\text{At}(\text{Flat}, \text{Axle})$ . Таким образом, взаимно исключаю-

щие отношения позволяют обнаруживать непосредственные конфликты, которые становятся следствием попыток поместить два объекта в одно и то же место одновременно.

После того как мы снова перейдем в начало цикла, на этот раз на уровне  $S_2$  будут присутствовать все литералы из цели и ни один из них не будет взаимно исключающим по отношению к любому другому. Это означает, что может существовать решение и в функции Extract-Solution будет предпринята попытка его найти. По сути, функция Extract-Solution решает булеву задачу CSP, переменными которой являются действия на каждом уровне, а значениями для каждой переменной служат индикаторы, показывающие, принадлежит ли она или не принадлежит к плану. Для этого можно воспользоваться стандартными алгоритмами CSP или определить функцию Extract-Solution как задачу поиска, в которой каждое состояние в поиске содержит указатель на уровень в графе планирования и множество невыполненных целей. Определим эту задачу поиска, как описано ниже.

- Первоначальным состоянием является последний уровень графа планирования,  $S_n$ , наряду с множеством целей из задачи планирования.
- Действия, доступные в любом состоянии на уровне  $S_i$ , должны выбирать любое бесконфликтное подмножество действий на уровне  $A_{i-1}$ , результаты которых покрывают цели в этом состоянии. Результирующее состояние имеет уровень  $S_{i-1}$  и включает в качестве своего множества целей все предусловия для выбранного множества действий. Под термином “бесконфликтный” подразумевается множество таких действий, что никакие два из них не являются взаимно исключающими и никакие два из их предусловий не являются взаимно исключающими.
- Задача состоит в том, чтобы достичь на уровне  $S_0$  такого состояния, чтобы все цели были выполнены.
- Стоимость каждого действия равна 1.

При решении данной конкретной задачи начнем с уровня  $S_2$ , где имеется цель  $\text{At}(\text{Spare}, \text{Axle})$ . Единственным вариантом, который может применяться для достижения этого множества целей, является  $\text{PutOn}(\text{Spare}, \text{Axle})$ . В результате мы переходим в состояние поиска на уровне  $S_1$  с целями  $\text{At}(\text{Spare}, \text{Ground})$  и  $\neg\text{At}(\text{Flat}, \text{Axle})$ . Первой цели можно достичь только с помощью действия  $\text{Remove}(\text{Spare}, \text{Trunk})$ , а последней — с помощью либо  $\text{Remove}(\text{Flat}, \text{Axle})$ , либо  $\text{LeaveOvernight}$ . Но действие  $\text{LeaveOvernight}$  является взаимно исключающим по отношению к  $\text{Remove}(\text{Spare}, \text{Trunk})$ , поэтому единственное решение состоит в том, чтобы выбрать  $\text{Remove}(\text{Spare}, \text{Trunk})$  и  $\text{Remove}(\text{Flat}, \text{Axle})$ . В результате мы переходим в состояние поиска на уровне  $S_0$  с целями  $\text{At}(\text{Spare}, \text{Trunk})$  и  $\text{At}(\text{Flat}, \text{Axle})$ . Обе из этих целей присутствуют в данном состоянии, поэтому налицо готовое решение: действия  $\text{Remove}(\text{Spare}, \text{Trunk})$  и  $\text{Remove}(\text{Flat}, \text{Axle})$  на уровне  $A_0$ , за которыми следует действие  $\text{PutOn}(\text{Spare}, \text{Axle})$  на уровне  $A_1$ .

Известно, что задача планирования является PSPACE-полной и что для построения графа планирования требуется полиномиальное время, поэтому в наихудшем случае может возникнуть ситуация, в которой извлечение решения окажется не осуществимым. Таким образом, потребуется определенное эвристическое руководство

при выборе среди действий в ходе обратного поиска. Одним из подходов, хорошо зарекомендовавших себя на практике, является жадный алгоритм, основанный на учете уровневой стоимости литералов. Для каждого набора целей применение этой эвристики осуществляется в описанном ниже порядке.

1. Определить литерал с максимальной уровневой стоимостью.
2. Чтобы достичь этого литерала, выбрать в первую очередь действие с самыми легкими для выполнения предусловиями. Это означает, что действие нужно выбирать так, чтобы сумма (или максимум) уровневых стоимостей его предусловий была минимальной.

### Завершение работы алгоритма Graphplan

До сих пор мы обходили проблему завершения работы алгоритма. Если задача не имеет решения, можно ли быть уверенными в том, что алгоритм Graphplan не будет проходить по циклу до бесконечности, расширяя граф планирования при каждой итерации? Ответ на этот вопрос является положительным, но полное доказательство такого утверждения выходит за рамки настоящей книги. Здесь мы кратко опишем основные идеи, особенно те, которые проливают свет на графы планирования в целом.

На первом этапе необходимо отметить, что характеристики некоторых свойств графов планирования монотонно увеличиваются или уменьшаются. Выражение “характеристика  $X$  увеличивается монотонно” означает, что множество экземпляров  $X$  на уровне  $i+1$  является надмножеством (необязательно строгим) множества на уровне  $i$ . Соответствующие свойства графов перечислены ниже.

- Количество литералов увеличивается монотонно. После того как некоторый литерал появляется на данном конкретном уровне, он будет появляться на всех последующих уровнях. Это связано с наличием сохраняющих действий; после того как литерал появляется, сохраняющие действия вызывают его сохранение навечно.
- Количество действий увеличивается монотонно. После того как некоторое действие появляется на данном конкретном уровне, оно будет появляться на всех последующих уровнях. Это — следствие увеличения количества литералов: если предусловия некоторого действия появляются на одном уровне, они будут появляться и на последующих уровнях и поэтому то же происходит и с действиями.
- Количество взаимно исключающих связей уменьшается монотонно. Если два действия на данном конкретном уровне  $A_i$  являются взаимно исключающими, то они должны также быть взаимно исключающими на всех предыдущих уровнях, на которых они появлялись вместе. То же утверждение остается справедливым и по отношению к взаимно исключающим связям между литералами. Это не означает, что взаимно исключающие связи характеризуются такими же особенностями и на рисунках с изображением графов планирования, поскольку на рисунках допускаются упрощения: на них не показывают ни литералов, которые не могут быть истинными на уровне  $S_i$ , ни действий, которые не могут быть выполнены на уровне  $A_i$ . Можно убедиться в справедливости утверждения, что “количество взаимно исключающих связей умень-

шается монотонно”, если учесть, что эти невидимые литералы и действия являются взаимно исключающими по отношению ко всем прочим.

Доказательство этих утверждений является довольно сложным, но может быть выполнено путем анализа отдельных случаев: если действия  $A$  и  $B$  являются взаимно исключающими на уровне  $A_1$ , то должны быть таковыми из-за наличия одного из трех типов взаимно исключающих связей. Первые два из них, несогласованные результаты и вмешательство, обусловлены свойствами самих действий, поэтому, если действия являются взаимно исключающими на уровне  $A_1$ , то будут взаимно исключающими на каждом уровне. Третий случай, с конкурирующими потребностями, зависит от условий на уровне  $S_1$ : этот уровень должен содержать предусловие действия  $A$ , которое является взаимно исключающим по отношению к предусловию действия  $B$ . Теперь отметим, что эти два предусловия могут быть взаимно исключающими, если они являются отрицаниями друг друга (и в этом случае они будут взаимно исключающими на каждом уровне) или все действия по достижении одного из них являются взаимно исключающими по отношению ко всем действиям, необходимым для достижения другого. Но мы уже знаем, что количество допустимых действий увеличивается монотонно, поэтому по индукции количество взаимно исключающих связей должно уменьшаться.

Поскольку количество действий и литералов увеличивается, а количество взаимно исключающих связей уменьшается, и поскольку имеется только конечное количество действий и литералов, каждый граф планирования в конечном итоге выравнивается — все последующие уровни в нем становятся идентичными. После того как график выровнялся, он может быть проанализирован, и если в нем отсутствует одна из целей задачи или две цели являются взаимно исключающими, то данная задача никогда не может быть решена, поэтому мы можем остановить работу алгоритма *Graphplan* и вернуть индикатор неудачи. Если же график выравнивается со всеми присутствующими и не взаимно исключающими целями, но функция *Extract-Solution* оказывается не в состоянии найти решения, то может потребоваться снова расширять график конечное количество раз, но так или иначе мы имеем возможность остановить работу алгоритма. Последний вариант завершения является более сложным и здесь не рассматривается.

## 11.5. ПЛАНИРОВАНИЕ С ПОМОЩЬЮ ПРОПОЗИЦИОНАЛЬНОЙ ЛОГИКИ

Как было описано в главе 10, планирование может осуществляться по принципу доказательства некоторой теоремы в рамках ситуационного исчисления. В подобной теореме утверждается, что при наличии начального состояния и аксиом состояния-предшественника, которые описывают результаты действий, цель будет истинной в ситуации, которая является результатом некоторой последовательности действий. В такой ранний период развития искусственного интеллекта, как 1969 год, данный подход считался слишком неэффективным для того, чтобы с его помощью можно было находить интересные планы. Проведенные в последнее время разработки в области эффективных алгоритмов формирования рассуждений для пропозициональной ло-

гики (см. главу 7) привели к возрождению интереса к планированию с помощью логических рассуждений.

Подход, принятый в данном разделе, основан на проверке **выполнимости** логического высказывания, а не на доказательстве теорем. Мы будем отыскивать модели для пропозициональных высказываний, которые выглядят примерно так:

*Начальное состояние*  $\wedge$  *Все возможные описания действий*  $\wedge$  *Цель*

Такое высказывание должно содержать пропозициональные символы, соответствующие каждому возможному проявлению действия; модель, в которой выполняется это высказывание, должна присваивать значение *true* действиям, являющимся частью правильного плана, и *false* — другим действиям. Присваивание, которое соответствует неправильному плану, не будет моделью, поскольку окажется несогласимым с утверждением, что цель истинна. Если задача планирования неразрешима, то данное высказывание будет невыполнимым.

### Описание задач планирования в пропозициональной логике

Процесс, который будет здесь применяться для перевода задач Strips на язык пропозициональной логики, представляет собой (так сказать) классический пример цикла представления знаний: мы начнем с множества аксиом, которое на первый взгляд кажется приемлемым, обнаружим, что эти аксиомы допускают появление фиктивных, не предусмотренных моделей, а затем запишем дополнительные аксиомы.

Начнем с очень простой задачи воздушной транспортировки. В начальном состоянии (время 0) самолет  $P_1$  находится в аэропорту *SFO*, а самолет  $P_2$  — в аэропорту *JFK*. Задача состоит в том, чтобы  $P_1$  находился в *JFK*, а  $P_2$  — в *SFO*; иными словами, самолеты должны поменяться местами. Прежде всего, потребуются отдельные пропозициональные символы для формирования утверждений о каждом временном этапе. Для обозначения временного этапа будут использоваться верхние индексы, как и в главе 7. Поэтому начальное состояние можно записать следующим образом:

$$\text{At}(P_1, \text{SFO})^0 \wedge \text{At}(P_2, \text{JFK})^0$$

(Напомним, что  $\text{At}(P_1, \text{SFO})^0$  — это атомарный символ.) Поскольку в пропозициональной логике не принято предположение о замкнутом мире, необходимо также определить высказывания, которые не являются истинными в начальном состоянии. Если же некоторые высказывания в начальном состоянии не известны, они могут оставаться неопределенными (**предположение об открытом мире**). В данном примере зададим следующее:

$$\neg\text{At}(P_1, \text{JFK})^0 \wedge \neg\text{At}(P_2, \text{SFO})^0$$

Сама цель должна быть связана с конкретным времененным этапом. Поскольку мы не знаем заранее, какое количество этапов потребуется для достижения цели, то можно попытаться сформулировать утверждение, что цель истинна в начальном состоянии, во время  $T=0$ . Это означает, что мы вводим утверждение  $\text{At}(P_1, \text{JFK})^0 \wedge \text{At}(P_2, \text{SFO})^0$ . Если эта попытка окажется неудачной, повторим ее снова во время  $T=1$  и т.д. до тех пор, пока не достигнем осуществимого плана с минимальной длиной. Для каждого значения  $T$  база знаний будет включать только высказывания, покрывающие временные этапы от 0 вплоть до  $T$ . Чтобы обеспечить завершение работы алгоритма, можно наложить произвольный верхний предел,

$T_{\max}$ . Этот алгоритм приведен в листинге 11.7. Альтернативный подход, позволяющий избежать использования многочисленных попыток решения, обсуждается в упр. 11.17.

**Листинг 11.7. Алгоритм SATplan.** Задача планирования преобразуется в высказывание в форме CNF, в котором подтверждается истинность цели на фиксированном временном этапе  $T$ , а аксиомы добавляются на каждом временном этапе вплоть до  $T$ . (Подробные сведения о преобразовании приведены в тексте главы.) Если алгоритм проверки выполнимости находит модель, то план извлекается путем поиска тех пропозициональных символов, которые относятся к действиям и которым в модели присвоено значение `true`. Если модель не существует, то процесс повторяется после передвижения цели на один этап дальше

---

```

function SATplan(problem,  $T_{\max}$ ) returns решение solution или индикатор
    неудачи failure
inputs: problem, задача планирования
         $T_{\max}$ , верхний предел длины плана

for  $T = 0$  to  $T_{\max}$  do
    cnf, mapping  $\leftarrow$  Translate-To-SAT(problem,  $T$ )
    assignment  $\leftarrow$  SAT-Solver(cnf)
    if присваивание assignment не является неопределенным then
        return Extract-Solution(assignment, mapping)
    return failure

```

---

Следующая проблема состоит в том, как закодировать описания действий в пропозициональной логике. Наиболее прямолинейный подход состоит в том, чтобы было предусмотрено по одному пропозициональному символу на каждое проявление действия; например, символ  $Fly(P_1, SFO, JFK)^0$  является истинным, если самолет  $P_1$  вылетает из аэропорта  $SFO$  в аэропорт  $JFK$  во время 0. Как и в главе 7, мы запишем пропозициональные версии аксиом состояния-преемника, разработанных для ситуационного исчисления в главе 10. Например, имеет место следующее:

$$At(P_1, JFK)^1 \Leftrightarrow (At(P_1, JFK)^0 \wedge \neg(Fly(P_1, JFK, SFO)^0 \wedge At(P_1, JFK)^0)) \\ \vee (Fly(P_1, SFO, JFK)^0 \wedge At(P_1, SFO)^0) \quad (11.1)$$

Это означает, что самолет  $P_1$  должен находиться в аэропорту  $JFK$  во время 1, если он был в  $JFK$  во время 0 и не улетел или если он был в аэропорту  $SFO$  во время 0 и прилетел в аэропорт  $JFK$ . Необходимо иметь по одной такой аксиоме для каждого самолета, аэропорта и временного этапа. Более того, каждый дополнительный аэропорт добавляет еще один способ путешествия из каждого конкретного аэропорта или в этот аэропорт и поэтому вносит еще больше дизъюнктов в правую часть каждой аксиомы.

После подготовки всех этих аксиом можно вызвать алгоритм проверки выполнимости, чтобы найти план. Это должен быть план, который достигает цели во время  $T=1$ , а именно план, в котором эти два самолета меняются местами. Теперь предположим, что база знаний представляет собой следующее высказывание:

$$\text{Начальное состояние} \wedge \text{Аксиомы состояния-преемника} \wedge \text{Цель}^1 \quad (11.2)$$

в котором утверждается, что цель истинна во время  $T=1$ . Читатель может проверить, что утверждение, в котором символы

$$Fly(P_1, SFO, JFK)^0 \text{ и } Fly(P_2, JFK, SFO)^0$$

являются истинными, а все другие символы действий являются ложными, представляет собой модель этой базы знаний. До сих пор все идет хорошо. Но есть ли другие возможные модели, которые способен вернуть этот алгоритм проверки выполнимости? Безусловно, да. Являются ли все эти другие модели удовлетворительными планами? К сожалению, нет. Рассмотрим довольно глупый план, заданный с помощью указанных символов действий:

$$\text{Fly}(\text{P}_1, \text{SFO}, \text{JFK})^0 \text{ и } \text{Fly}(\text{P}_1, \text{JFK}, \text{SFO})^0 \text{ и } \text{Fly}(\text{P}_2, \text{JFK}, \text{SFO})^0$$

Этот план глуп потому, что самолет  $P_1$  вначале находится в аэропорту  $SFO$ , поэтому действие  $\text{Fly}(\text{P}_1, \text{JFK}, \text{SFO})^0$  является неосуществимым. Тем не менее этот план действительно представляет собой модель высказывания из уравнения 11.2! Иными словами, он согласуется со всем, чтобы сказано до сих пор об этой задаче. Чтобы понять, чем может быть вызвано появление такого плана, мы должны проанализировать более тщательно, что сказано в аксиомах состояния-преемника (таких как уравнение 11.1) о действиях, предусловия которых не выполнены. Аксиомы правильно предсказывают, что ничего не произойдет при попытке выполнить подобное действие (см. упр. 11.15), но в них ничего не сказано, что такое действие не может быть выполнено! Чтобы предотвратить выработку планов с недопустимыми действиями, мы должны добавить **аксиомы предусловий** (precondition axiom), которые указывают, что для совершения любого действия требуется, чтобы были выполнены его предусловия<sup>6</sup>. Например, требуется указать следующее:

$$\text{Fly}(\text{P}_1, \text{JFK}, \text{SFO})^0 \Rightarrow \text{At}(\text{P}_1, \text{JFK})^0$$

Поскольку указано, что в начальном состоянии символ  $\text{At}(\text{P}_1, \text{JFK})^0$  является ложным, эта аксиома гарантирует, что в каждой модели символу  $\text{Fly}(\text{P}_1, \text{JFK}, \text{SFO})^0$  также будет присвоено ложное значение. После введения аксиом предусловия существует одна и только одна модель, которая удовлетворяет всем аксиомам, если цель должна быть достигнута во время 1, а именно модель, в которой самолет  $P_1$  летит в аэропорт  $JFK$ , а самолет  $P_2$  — в аэропорт  $SFO$ . Обратите внимание на то, что это решение обеспечивает выполнение двух параллельных действий, как и при использовании алгоритма Graphplan или POP.

Неожиданности возникают после добавления третьего аэропорта,  $LAX$ . Теперь для каждого самолета имеются два действия, которые являются допустимыми в каждом состоянии. После применения алгоритма проверки выполнимости мы обнаружим, что модель с символами  $\text{Fly}(\text{P}_1, \text{SFO}, \text{JFK})^0$ , а также  $\text{Fly}(\text{P}_2, \text{JFK}, \text{SFO})^0$  и  $\text{Fly}(\text{P}_2, \text{JFK}, \text{LAX})^0$  удовлетворяет всем аксиомам. Это означает, что аксиомы состояния-преемника и аксиомы предусловия разрешают самолету вылететь в два аэропорта назначения одновременно! Предусловия для двух полетов самолета  $P_2$  выполнены в начальном состоянии; аксиомы состояния-преемника указывают, что самолет  $P_2$  должен находиться и в аэропорту  $SFO$ , и в аэропорту  $LAX$  во время 1, поэтому цель выполнена. Очевидно, что необходимо ввести дополнительные аксиомы для устранения таких фиктивных решений. Один из подходов состоит в том, что требуется ввести **аксиомы исключения действий** (action exclusion axiom), которые предотвращают одновременное выполнение несовместимых действий. Например, можно принудительно ввести полное исключение, добавив все возможные аксиомы в такой форме:

---

<sup>6</sup> Обратите внимание на то, что добавление аксиом предусловий означает, что не нужно будет включать предусловия для действий в аксиомы состояния-преемника.

$$\neg(Fly(P_2, JFK, SFO)^0 \wedge Fly(P_2, JFK, LAX)^0)$$

Эти аксиомы гарантируют, что никакие два действия не могут происходить одновременно. Они устранит все фиктивные планы, но приведут также к полному упорядочению каждого плана. В результате гибкость частично упорядоченных планов будет потеряна; кроме того, в результате увеличения количества временных этапов в плане может увеличиться продолжительность вычислений.

Вместо полного исключения можно потребовать только частичного исключения, т.е. предотвращения одновременных действий, только если они мешают друг другу. Применяемые при этом условия являются аналогичными условиям для взаимно исключающих действий: два действия не могут происходить одновременно, если одно из них отрицает предусловие или результат другого. Например, не могут происходить одновременно действия  $Fly(P_2, JFK, SFO)^0$  и  $Fly(P_2, JFK, LAX)^0$ , поскольку каждое из них отрицает предусловие другого; с другой стороны, действия  $Fly(P_1, SFO, JFK)^0$  и  $Fly(P_2, JFK, SFO)^0$  могут осуществляться одновременно, поскольку в них два самолета не мешают друг другу. Частичное исключение позволяет устранять фиктивные планы, не вводя принудительно полное упорядочение.

Аксиомы исключения иногда выглядят как довольно слепой инструмент. Поэтому вместо формирования утверждения о том, что ни один самолет не может вылететь в два аэропорта одновременно, можно просто выдвинуть требование, чтобы ни один объект не мог находиться в двух местах одновременно:

$$\forall p, x, y, t \ x \neq y \Rightarrow \neg(At(p, x)^t \wedge At(p, y)^t)$$

Из этого факта, применяемого в сочетании с аксиомами состояния-преемника, следует, что самолет не может вылететь в два аэропорта одновременно. Факты, подобные этому, называются **ограничениями состояния** (state constraint). Безусловно, в пропозициональной логике необходимо будет записать все базовые экземпляры каждого ограничения состояния. Для задачи с аэропортами этого ограничения состояния будет достаточно, чтобы исключить все фиктивные планы. Ограничения состояния часто являются гораздо более компактными по сравнению с аксиомами исключения действия, но их не всегда можно легко вывести из первоначального описания задачи на языке Strips.

Подводя итог, отметим, что планирование в форме доказательства выполнимости предусматривает поиск моделей для высказывания, включающего описание начального состояния, цели, аксиом состояния-преемника, аксиом предусловий, а также либо аксиом исключения действия, либо аксиом ограничения состояния. Можно показать, что эта коллекция аксиом является достаточной, в том смысле, что при их использовании больше не возникают какие-либо фиктивные “решения”. Любая модель, в которой выполняется такое пропозициональное высказывание, будет представлять собой действительный план для первоначальной задачи; это означает, что любая линеаризация этого плана будет представлять собой допустимую последовательность действий, которая позволяет достичь цели.

### Сложности, связанные с использованием пропозициональных кодировок

Основным недостатком описанного пропозиционального подхода являются колоссальные размеры пропозициональной базы знаний, которая формируется на основе первоначальной задачи планирования. Например, схема действий  $Fly(p, a_1, a_2)$  пре-

образуется в  $T \times |Planes| \times |Airports|^2$  различных пропозициональных символов. Вообще говоря, общее количество символов действий ограничено значением  $T \times |Act| \times |O|^P$ , где  $|Act|$  — количество схем действий;  $|O|$  — количество объектов в проблемной области;  $P$  — максимальная арность (количество параметров) любой схемы действий. Количество выражений еще больше. Например, при 10 временных этапах, 12 самолетах и 30 аэропортах полная аксиома исключения действий состоит из 583 миллионов выражений.

Поскольку количество символов действий экспоненциально зависит от арности схемы действий, одним из способов преодоления указанного недостатка может оказаться попытка уменьшить арность. Это можно сделать, заимствовав одну идею из области семантических сетей (см. главу 10). В семантических сетях используются только бинарные предикаты; предикаты с большим количеством параметров сводятся к множеству бинарных предикатов, которые описывают каждый параметр отдельно. Применяя эту идею к символу действий, такому как  $Fly(P_1, SFO, JFK)^0$ , получим следующие три новых символа:

- $Fly_1(P_1)^0$ : самолет  $P_1$  прилетел во время 0
- $Fly_2(SFO)^0$ : аэропортом отправления в этом полете был  $SFO$
- $Fly_3(JFK)^0$ : аэропортом назначения в этом полете был  $JFK$

Этот процесс, называемый **расщеплением символов** (symbol splitting), позволяет устраниТЬ необходимость в использовании экспоненциального количества символов. Теперь требуется только  $T \times |Act| \times P \times |O|$  символов.

Расщепление символов само по себе позволяет сократить количество символов, но не приводит к автоматическому уменьшению количества аксиом в базе знаний. Это означает, что если бы каждый символ действия в каждом выражении был просто заменен конъюнкцией трех символов, то общий размер базы знаний остался бы примерно тем же самым. Расщепление символов фактически приводит к уменьшению базы знаний потому, что некоторые из расщепленных символов станут нерелевантными для определенных аксиом и могут быть удалены. Например, рассмотрим аксиому состояния-преемника, приведенную в уравнении 11.1, модифицированную так, чтобы в нее был включен символ аэропорта  $LAX$  и исключены предусловия действия (которые будут учитываться с помощью отдельных аксиом предусловия):

$$\begin{aligned} At(P_1, JFK)^1 \Leftrightarrow & (At(P_1, JFK)^0 \wedge \neg Fly(P_1, JFK, SFO)^0 \wedge \neg Fly(P_1, JFK, LAX)^0) \\ & \vee Fly(P_1, SFO, JFK)^0 \vee Fly(P_1, LAX, JFK)^0 \end{aligned}$$

В первом условии сказано, что самолет  $P_1$  должен быть в аэропорту  $JFK$ , если он находился в нем во время 0 и не улетел из  $JFK$  в какой-то другой город, неважно, какой именно; во втором условии сказано, что он должен находиться в аэропорту  $JFK$ , если он прилетел туда из другого города, неважно, из какого именно. При использовании этих расщепленных символов мы можем удалить параметр, значение которого нас не интересует:

$$\begin{aligned} At(P_1, JFK)^1 \Leftrightarrow & (At(P_1, JFK)^0 \wedge \neg(Fly_1(P_1)^0 \wedge Fly_2(JFK)^0)) \\ & \vee (Fly_1(P_1)^0 \wedge Fly_3(JFK)^0) \end{aligned}$$

Обратите внимание на то, что в этой аксиоме аэропорты  $SFO$  и  $LAX$  больше не упоминаются. Вообще говоря, теперь расщепленные символы действий позволяют добиться того, чтобы размер каждой аксиомы состояния-преемника был независимым от количества аэропортов. Аналогичные сокращения допускаются в аксиомах предусловия и ак-

сиомах исключения действий (см. упр. 11.16). Для описанного выше случая с 10 временными этапами, 12 самолетами и 30 аэропортами полная аксиома исключения действий сокращается с 583 миллионов выражений до 9360 выражений.

Но остается непреодоленным еще один недостаток: представление с помощью расщепленных символов не допускает описания параллельных действий. Рассмотрим два параллельных действия,  $Fly(P_1, SFO, JFK)^0$  и  $Fly(P_2, JFK, SFO)^0$ . Преобразовав их в расщепленные представления, получим следующее:

$$\begin{aligned} Fly_1(P_1)^0 \wedge Fly_2(SFO)^0 \wedge Fly_3(JFK)^0 \wedge \\ Fly_1(P_2)^0 \wedge Fly_2(JFK)^0 \wedge Fly_3(SFO)^0 \end{aligned}$$

Теперь больше нет возможности определить с помощью этого выражения, что произошло! Мы знаем, что самолеты  $P_1$  и  $P_2$  вылетели, но не можем выяснить места их отправления и назначения. Это означает, что должна использоваться полная аксиома исключения действий со всеми указанными выше недостатками.

Планировщики, основанные на проверке выполнимости, способны обрабатывать крупные задачи планирования, например находить оптимальные тридцатистадийные решения для задач планирования в мире блоков с десятками блоков. Размер пропозиционального представления и стоимость решения в высшей степени зависят от задачи, но в большинстве случаев узким местом становится нехватка памяти, требуемой для хранения пропозициональных аксиом. Одним из интересных результатов этих исследований оказалось то, что алгоритмы поиска с возвратами, такие как DPLL, часто лучше решают задачи планирования по сравнению с алгоритмами локального поиска, подобными WalkSAT. Это связано с тем, что основная часть пропозициональных аксиом представляет собой хорновские выражения, которые эффективно обрабатываются с помощью метода распространения единичных выражений. Это наблюдение привело к разработке гибридных алгоритмов, в которых комбинируется некоторый метод случайного поиска с возвратами и метод распространения единичных выражений.

## 11.6. АНАЛИЗ РАЗЛИЧНЫХ ПОДХОДОВ К ПЛАНИРОВАНИЮ

Планирование — это область искусственного интеллекта, которая в настоящее время привлекает значительный интерес. Одна из причин такой ситуации состоит в том, что в планировании объединяются два основных направления развития искусственного интеллекта, которые рассматривались нами до сих пор, — поиск и логика. Это означает, что любой планировщик может рассматриваться либо как программа, в которой осуществляется поиск решения, либо как такая программа, которая (конструктивно) доказывает существование решения. Такое перекрестное обогащение идеями, взятыми из этих двух областей, привело не только к повышению производительности, которое за последнее десятилетие достигло нескольких порядков величины, но и к расширению использования планировщиков в производственных приложениях. К сожалению, у нас еще нет четкого понимания того, какие методы являются в наибольшей степени подходящими для задач того или иного типа. Вполне возможно также, что появятся новые методы, которые вытеснят существующие.

Планирование главным образом представляет собой такое занятие, в котором приходится держать под контролем комбинаторный взрыв. Если в проблемной об-

ласти имеется  $p$  примитивных высказываний, то количество состояний становится равным  $2^p$ . В сложных проблемных областях величина  $p$  может стать весьма значительной. Следует также учитывать, что объекты в проблемной области характеризуются не только свойствами (*Location*, *Color* и т.д.), но и отношениями (*At*, *On*, *Between* и т.д.). При наличии  $d$  объектов в проблемной области с тернарными отношениями количество состояний достигает  $2^{d^3}$ . На этом основании можно сделать вывод, что в наихудшем случае попытка решить задачу планирования становится безнадежной.

Мощным средством преодоления подобной пессимистической ситуации становится подход по принципу “разделяй и властвуй”. В наилучшем случае (при полной декомпонуемости задачи) подход “разделяй и властвуй” обеспечивает экспоненциальное ускорение работы. Однако декомпонуемость нарушается из-за отрицательных взаимодействий между действиями. Планировщики с частичным упорядочением позволяют справиться с такой ситуацией с помощью причинных связей — мощного подхода к формированию представлений, но, к сожалению, каждый конфликт должен быть разрешен с помощью выбора определенного варианта (связанного с размещением конфликтующего действия до или после связи), а количество таких вариантов может увеличиваться экспоненциально. Алгоритм Graphplan позволяет избежать необходимости использования этих вариантов на этапе формирования графа, поскольку в нем для регистрации конфликтов используются взаимно исключающие связи, а выбор варианта, касающегося того, как должны быть разрешены эти конфликты, фактически не осуществляется. Алгоритм SATplan предоставляет аналогичный набор взаимно исключающих отношений, но в нем для этого применяется общая форма CNF, а не специальная структура данных. Насколько успешным окажется такой подход, зависит от области применения решателя задач SAT.

Иногда возможность эффективного решения задачи обеспечивается путем определения того, какие отрицательные взаимодействия могут быть исключены. Задача называется имеющей **упорядочиваемые подцели**, если существует такой порядок подцелей, что планировщик может достичь их в указанном порядке, не будучи вынужденным отменять какую-либо из ранее достигнутых подцелей. Например, если в мире блоков цель состоит в построении столбика (например, в котором блок *A* стоит на *B*, который, в свою очередь, стоит на *C*, стоящем, в свою очередь, на столе, *Table*), то подцели являются упорядочиваемыми снизу вверх: если вначале будет достигнута подцель “*C* на *Table*”, то никогда не придется ее отменять в ходе достижения других подцелей. Планировщик, в котором используется такой прием с упорядочением снизу вверх, способен решить любую задачу в проблемной области мира блоков без возвратов (хотя и может оказаться, что он не всегда находит самый короткий план).

В качестве более сложного примера укажем, что для планировщика Remote Agent, который управляем космическим аппаратом Deep Space One агентства NASA, было определено, что высказывания, касающиеся управления космическим аппаратом, являются упорядочиваемыми. По-видимому, в этом нет ничего удивительного, поскольку космический аппарат проектируется его разработчиками так, чтобы была возможность управлять им как можно проще (разумеется, с учетом всех прочих ограничений). Пользуясь преимуществом последовательного упорядочения целей, планировщик Remote Agent был способен в процессе планирования устранять ос-

новную часть поиска. Это означает, что он оказался достаточно быстродействующим для того, чтобы с его помощью можно было управлять этим космическим аппаратом в реальном времени, а такая задача раньше казалась невыполнимой.

Существует несколько способов ограничения комбинаторного взрыва. Как было показано в главе 5, есть несколько методов управления возвратами в задачах удовлетворения ограничений (Constraint Satisfaction Problem — CSP), таких как возвраты, управляемые зависимостями. Все эти методы могут применяться и в планировании. Например, задачу извлечения решения из графа планирования можно сформулировать как булеву задачу CSP, переменные которой указывают, должно ли данное конкретное действие произойти в данное конкретное время. Эту задачу CSP можно решить с использованием любого из алгоритмов, приведенных в главе 5, такого как алгоритм с минимальными конфликтами. Тесно связанный с этим метод, используемый в системе Blackbox, состоит в преобразовании графа планирования в выражение CNF, а затем извлечении плана с использованием решателя задач SAT. По-видимому, такой подход обладает более высокой производительностью, чем SATplan, и причиной этого, скорее всего, является то, что в графе планирования уже устранены многие невозможные состояния и действия из рассматриваемой задачи. Кроме того, такой подход действует лучше по сравнению с алгоритмом Graphplan, по-видимому, из-за того, что поиск условий выполнимости, подобный алгоритму WalkSAT, характеризуется гораздо большей гибкостью, чем ограниченный поиск с возвратами, используемый в алгоритме Graphplan.

Нет никакого сомнения в том, что такие планировщики, как Graphplan, SATplan и Blackbox, привели к значительному прогрессу в области планирования, поскольку позволили, во-первых, поднять уровень производительности систем планирования, а во-вторых, дали возможность понять связанные с этим представительные и комбинаторные проблемы. Однако эти методы по своей сути являются пропозициональными и поэтому предназначены исключительно для использования в тех проблемных областях, которые они способны представить. (Например, задачи планирования экспедиторской доставки с несколькими десятками объектов и местонахождений могут потребовать гигабайтов памяти для хранения соответствующих выражений CNF.) Создается впечатление, что для достижения дальнейшего прогресса в этой области потребуются представления и алгоритмы в логике первого порядка, хотя такие структуры, как графы планирования, по-прежнему будут оставаться полезным источником эвристик.

## 11.7. РЕЗЮМЕ

В этой главе определена задача планирования в детерминированных, полностью наблюдаемых вариантах среды. В ней описаны основные представления, используемые для задач планирования, и представлено несколько алгоритмических подходов к их решению. Наиболее важные понятия, изложенные в этой главе, перечислены ниже.

- Системы планирования основаны на использовании алгоритмов решения задач, которые применяются к явным представлениям состояний и действий в пропозициональной логике (или логике первого порядка). Такие представления обеспечивают возможность получения эффективных эвристик, а также разработки мощных и гибких алгоритмов для решения задач планирования.

- В языке Strips применяются описания действий в терминах их предусловий и результатов, а также описания начальных и целевых состояний в виде конъюнкций положительных литералов. В языке ADL некоторые ограничения языка Strips ослаблены и допускается использование дизъюнкции, отрицания и кванторов.
- Поиск в пространстве состояний может действовать в прямом (**прогрессивном**) направлении или в обратном (**ретрессивном**) направлении. Эффективные эвристики могут быть получены путем принятия предположения о независимости подцелей, а также с помощью различных ослаблений задачи планирования.
- В алгоритмах планирования с частичным упорядочением (Partial-Order Planning — POP) пространство планов исследуется без стремления к созданию полностью упорядоченной последовательности действий. Такие алгоритмы действуют в обратном направлении от цели, добавляя в план действия, нужные для достижения каждой подцели. Такой подход становится особенно эффективным при решении задач, приемлемых для использования подхода по принципу “разделяй и властвуй”.
- **Граф планирования** может формироваться инкрементно, начиная с начального состояния. Каждый его уровень содержит надмножество всех литералов или действий, которые могут обнаруживаться на данном временном этапе. Кроме того, на каждом уровне условно задаются отношения взаимного исключения, или **взаимно исключающие отношения** между литералами или действиями, которые не могут происходить одновременно. Графы планирования позволяют получать полезные эвристики для планировщиков в пространстве состояний и планировщиков с частичным упорядочением и могут использоваться непосредственно в алгоритме Graphplan.
- Алгоритм Graphplan обрабатывает граф планирования, используя обратный поиск для извлечения плана. Этот алгоритм допускает определенное частичное упорядочение между действиями.
- В алгоритме SATplan задача планирования преобразуется в пропозициональные аксиомы и после этого к ним применяется алгоритм проверки выполнимости для поиска модели, соответствующей действительному плану. Было разработано несколько разных пропозициональных представлений, обладающих различной степенью компактности и эффективности.
- Каждый из основных подходов к планированию имеет своих сторонников, и еще не достигнуто полное согласие в том, какой из этих подходов является наилучшим. Конкуренция между этими подходами и их взаимное обогащение привели к существенному повышению эффективности систем планирования.

---

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Направление планирования в области искусственного интеллекта сформировалось на основе исследований в части поиска в пространстве состояний, доказательства теорем и теории управления, а также на основании практических потребностей робототехники, составления расписаний и других проблемных областей. Первой

важной системой планирования стала система Strips [466], которая наглядно иллюстрирует продуктивность взаимодействия этих научных направлений. Система Strips была разработана как планирующий компонент программного обеспечения для проекта создания робота Shakey в институте SRI. Модель ее общей структуры управления была создана на основе программы GPS (General Problem Solver — общий решатель задач) [1129] — системы поиска в пространстве состояний, в которой используется анализ целей и средств (means—ends analysis). В системе Strips применялась одна из версий системы доказательства теорем QA3 [592] в качестве процедуры определения истинности предусловий для действий. Точные определения для языка Strips и анализ этого языка представлены Лифшицем [928]. Байлендер [213] показал, что простые задачи планирования Strips являются PSPACE-полными. В [467] приведена историческая ретроспектива проекта Strips и дан краткий обзор того, как этот проект связан с более современными разработками в области планирования.

Способ представления действий, использовавшийся в системе Strips, оказал гораздо более значительное влияние на дальнейшие разработки, чем ее алгоритмический подход. С тех пор почти во всех системах планирования применяется тот или иной вариант языка Strips. К сожалению, из-за огромного разнообразия вариантов задача их сравнения стала чрезмерно трудной. Со временем возникло лучшее понимание ограничений и компромиссов между формальными подходами. В языке ADL (Action Description Language — язык описания действий) [1195] ослаблены некоторые ограничения языка Strips и создана возможность представлять более реалистичные задачи. В [1119] рассматриваются схемы, применимые для компиляции определений ADL в определения Strips. Для использования в качестве стандартизированного синтаксиса, допускающего синтаксический анализ с помощью компьютера, который предназначен для представления определений на языках Strips, ADL и других языках, был предложен язык PDDL (Problem Domain Description Language — язык описания проблемной области) [548]. PDDL использовался в качестве стандартного языка для соревнований по планированию на конференции AIPS, начиная с 1998 года.

В первой половине 1970-х годов планировщики в основном применялись для получения полностью упорядоченных последовательностей действий. Декомпозиция задачи осуществлялась путем вычисления субплана для каждой подцели с последующей увязкой субпланов в единую цепочку в некотором порядке. Вскоре было обнаружено, что такой подход, названный **линейным планированием** в [1337], является неполным. Он не позволяет решать некоторые очень простые задачи, такие как аномалия Зюссмана (см. упр. 11.11), обнаруженная Алленом Брауном во время экспериментов с системой Hacker [1474]. Полный планировщик должен обеспечивать **чередование** действий из различных субпланов в пределах единой последовательности. Понятие упорядочиваемых подцелей [837] точно соответствует множеству задач, для которых планировщики без чередования позволяют получить полное решение.

Одним из решений проблемы чередования оказалось планирование с регрессией от цели. Это — метод, в котором этапы полностью упорядоченного плана переупорядочиваются так, чтобы можно было избежать конфликта между подцелями. Данный подход был предложен Уолдингером [1551], кроме того, использовался в системе Warplan Уоррена [1554]. Система Warplan замечательна также тем, что была первым планировщиком, написанным на языке логического программирования (Prolog), и остается одним из лучших примеров невероятной экономии объема кода,

которая иногда может быть достигнута с использованием логического программирования: программа Warplan состоит только из 100 строк кода, что составляет лишь небольшую часть размера сравнимых с ней планировщиков на то время. Система Interplan [1493], [1494] обеспечивала также произвольное чередование этапов плана, что дало возможность преодолеть аномалию Зюссмана и связанные с ней проблемы.

К основным идеям, лежащим в основе планирования с частичным упорядочением, относятся обнаружение конфликтов [1493] и защита достигнутых условий от вмешательства [1474]. Первыми примерами средств создания планов с частичным упорядочением (которые в то время назывались **сетями задач**) были планировщик Noah Сакердоти [1337], [1338] и система Nonlin Тейта [1494], [1495]<sup>7</sup>.

Планирование с частичным упорядочением в течение следующих 20 лет стало ведущим направлением разработок, несмотря на то, что в течение основной части этого периода специфика данной области не нашла широкого понимания. Система Tweak Чепмена [234] стала образцом логической реконструкции и упрощения работ по планированию, проводимых в то время; формулировки, используемые в этой системе, были достаточно ясными для того, чтобы можно было доказывать полноту и разрешимость (либо NP-трудность и неразрешимость) различных формулировок задач планирования. Эта работа Чепмена привела к созданию Макаллестером и Розенблиттом [1008] полного планировщика с частичным упорядочением, который впервые можно было вполне обоснованно считать имеющим достаточно простое и доступное для чтения описание [1008]. Одна из реализаций алгоритма Макаллестера и Розенблитта, разработанная Содерлендом и Уэллом и получившая название SNLP [1444], нашла широкое распространение и впервые позволила многим исследователям изучать и проводить эксперименты по планированию с частичным упорядочением. Алгоритм POP, описанный в этой главе, основан на алгоритме SNLP.

Кроме того, группа Уэлда разработала UCPOP [1202], первый планировщик для задач, представленных на языке ADL. В планировщике UCPOP применяется эвристика, в которой учитывается количество невыполненных целей. Применяемый в нем алгоритм работал немного быстрее, чем SNLP, но редко оказывался способным находить планы, состоящие больше чем примерно из десяти этапов. Хотя для планировщика UCPOP были разработаны усовершенствованные эвристики [543], [751], направление планирования с частичным упорядочением постепенно вытеснялось из перспективных исследований в 1990-х годах по мере появления более быстрых методов. Но в [1135] было показано, что эта область исследований вполне заслуживает возобновления к ней интереса: предложенный в этой работе планировщик RePOP при использовании точной эвристики, полученной из графа планирования, оказался намного более масштабируемым по сравнению с планировщиком Graphplan и сумел составить конкуренцию самым быстрым планировщикам в пространстве состояний.

Работы Авrimа Блюма и Меррика Фурста [139], [140] способствовали дальнейшему пробуждению интереса к этой области планирования, поскольку созданная ими система Graphplan оказалась на несколько порядков быстрее по сравнению с планировщиками с частичным упорядочением, применявшимися в то

<sup>7</sup> В терминологии существует некоторая путаница. Многие авторы используют термин **нелинейный** (nonlinear) вместо “частично упорядоченный” (partially ordered). Такое употребление терминов немного отличается от первоначального употребления, которое в работах Сакердоти относилось к чередующимся планам.

время. За ней вскоре последовали разработки других систем с графами планирования, такие как IPP [813], Stan [491] и SGP [1569]. Структура данных, весьма напоминающая граф планирования, была разработана немного раньше Галлабом и Ларуэлем [549], которые использовали такую структуру данных в планировщике с частичным упорядочением IXTeT для получения точных эвристик, применяемых для управления поиском. В [1136] приведен очень подробный анализ эвристик, полученных на основе графов планирования. Приведенное в этой главе описание графов планирования главным образом основано на этой работе и на конспектах лекций, подготовленных Суббарао Камбхампати (Subbarao Kambhampati). Как уже указывалось в этой главе, граф планирования позволяет управлять поиском решения задачи с помощью многих способов. Например, победителем соревнования по планированию на конференции AIPS 2002 года стал алгоритм LPG [544], в котором осуществляется поиск в графах планирования с использованием метода локального поиска, основанного на алгоритме WalkSAT.

Подход к планированию как проверке выполнимости и алгоритм SATplan были предложены Каутцем и Селманом [778], которых натолкнул на эту идею поразительный успех в использовании жадного локального поиска для решения задач проверки выполнимости (см. главу 7). Кроме того, Каутц и др. [777] исследовали различные формы пропозициональных представлений для аксиом Strips и обнаружили, что наиболее компактные формы не обязательно способствуют достижению наименьшей продолжительности вычисления решения. Систематический анализ был проведен Эрнстом и др. [442], которые разработали также автоматический “компилятор” для формирования пропозициональных представлений из формулировок задач на языке PDDL. Планировщик Blackbox, в котором объединены идеи алгоритмов Graphplan и SATplan, был разработан Каутцем и Селманом [779].

Повторное пробуждение интереса к планированию в пространстве состояний было в основном вызвано появлением программы UnPOP, разработанной Макдермоттом [1024], который впервые предложил эвристику с учетом расстояния, основанную на ослабленной задаче с исключенными списками удаления. Само название UnPOP (отказ от POP) стало реакцией на чрезмерное в то время сосредоточение работ на планировании с частичным упорядочением; Макдермотт считал, что другие подходы не получают того внимания, которого они заслуживают. В алгоритме HSP (Heuristic Search Planner — планировщик с эвристическим поиском) Бонета и Геффнера [147] и его разработанных позже аналогах впервые удалось добиться практического применения поиска в пространстве состояний при решении крупных задач планирования. В то время самым успешным алгоритмом поиска в пространстве состояний был алгоритм FastForward, или FF, Хоффмана [667], который стал победителем в соревновании по планированию на конференции AIPS в 2000 году. В алгоритме FF используется упрощенная эвристика для графа планирования в сочетании с очень быстрым алгоритмом поиска, представляющим собой новаторское объединение прямого и локального поиска.

В дальнейшем появился интерес к использованию представления планов в виде **бинарных диаграмм решения** (binary decision diagram — BDD) — компактного описания конечных автоматов, которое широко исследовалось в сообществе разработчиков средств проверки аппаратного обеспечения [266], [1031]. Существуют методы доказательства свойств бинарных диаграмм решения, включая свойство быть решением задачи планирования. В [261] представлен планировщик, основанный на этом

подходе. Применялись также другие представления; например, в [1549] приведен обзор использования целочисленного программирования для планирования.

Точки над “*i*” в этом вопросе еще окончательно не расставлены, но уже появились некоторые интересные сопоставления различных подходов к планированию. В [646] проанализировано несколько классов задач планирования и показано, что подходы на основе ограничений, такие как Graphplan и SATplan, являются наилучшими для NP-трудных проблемных областей, а подходы на основе поиска лучше подходят для проблемных областей, в которых приемлемые решения могут быть найдены без поиска с возвратами. Алгоритмы Graphplan и SATplan сталкиваются с затруднениями при использовании в проблемных областях со многими объектами, поскольку наличие большого количества объектов означает, что в этих алгоритмах приходится создавать много действий. В некоторых случаях возникновение такой проблемы можно отсрочить или вообще ее избежать, вырабатывая пропозиционализированные действия динамически, по мере необходимости, а не конкретизируя их все до начала поиска.

В [1567], [1568] приведены два превосходных обзора современных алгоритмов планирования. Любопытно наблюдать за тем, какие изменения произошли за пять лет, которые прошли за время между этими двумя обзорами: первый из них сосредотачивался на планировании с частичным упорядочением, а во втором были представлены алгоритмы Graphplan и SATplan. В книге *Readings in Planning* [19] приведена всеобъемлющая антология многих из самых лучших ранних статей в этой области, включая несколько хороших обзоров. В [1629] приведен обзор методов планирования с частичным упорядочением, который занимает целую книгу.

Исследования по планированию играли центральную роль в искусственном интеллекте со времени появления этого научного направления, а статьи по планированию занимают основной объем ведущих журналов и материалов конференций по искусственному интеллекту. Проводятся также специализированные конференции по планированию, такие как *International Conference on AI Planning Systems* (AIPS), *International Workshop on Planning and Scheduling for Space* и *European Conference on Planning*.

## УПРАЖНЕНИЯ

- 11.1.** Опишите различие и сходство между решением задач и планированием.
- 11.2.** При наличии аксиом, приведенных в листинге 11.1, определите все применимые конкретные экземпляры действия  $Fly(p, from, to)$  в состоянии, описанном следующим выражением:
 
$$\begin{aligned} & At(P_1, JFK) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \\ & \wedge Airport(JFK) \wedge Airport(SFO) \end{aligned}$$
- 11.3.** Рассмотрим, как можно было бы преобразовать множество схем Strips в аксиомы состояния-преемника ситуационного исчисления (см. главу 10).
  - Изучите схему для действия  $Fly(p, from, to)$ . Запишите логическое определение для предиката  $FlyPrecond(p, from, to, s)$ , который является

истинным, если предусловия для действия  $Fly(p, from, to)$  выполнены в ситуации  $s$ .

- Затем, предположив, что  $Fly(p, from, to)$  является единственной схемой действия, доступной для агента, запишите аксиому состояния-преемника для литерала  $At(p, x, s)$ , который представляет ту же информацию в виде схемы действия.
- Теперь предположим, что есть еще один метод перемещения в пространстве — телепортация:  $Teleport(p, from, to)$ . Он имеет дополнительное предусловие  $\neg Warped(p)$  (пространство не искривлено) и дополнительный результат  $Warped(p)$  (пространство искривлено). Объясните, как должна быть модифицирована база знаний ситуационного исчисления.
- Наконец, разработайте общую и точно заданную процедуру для осуществления преобразования из множества схем Strips в множество аксиом состояния-преемника.

**11.4.** Лабораторная обезьяна сталкивается с задачей “обезьяна и бананы”, в которой с потолка свисают бананы, не достижимые непосредственно. Имеется ящик, позволяющий обезьяне достать бананы, если она на него заберется. Первоначально обезьяна находится в позиции  $A$ , бананы — в позиции  $B$ , а ящик — в позиции  $C$ . Обезьяна и ящик имеют высоту  $Low$ , но после того как обезьяна заберется на ящик, она будет иметь высоту  $High$ , такую же, как и у бананов. Действия, доступные для обезьяны, включают  $Go$  (Переход с одного места в другое),  $Push$  (Перемещение объекта из одного места в другое),  $ClimbUp$  (Подъем на объект) или  $ClimbDown$  (Спуск с объекта), а также  $Grasp$  (Схватывание объекта) или  $Ungrasp$  (Отпускание объекта). Схватывание приводит к овладению объектом, если обезьяна и объект находятся в одном и том же месте и на одной и той же высоте.

- a) Составьте начальное описание состояния.
- b) Запишите определения указанных шести действий в стиле Strips.
- c) Предположим, что обезьяна хочет поставить в тупик ученых (которые перестали за ней следить и отправились пить чай), схватив бананы, но оставив ящик на первоначальном месте. Запишите эту задачу как общую цель (т.е. не предполагая, что ящик обязательно находится в позиции  $C$ ) на языке ситуационного исчисления. Может ли задача достижения этой цели быть решена с помощью системы в стиле Strips?
- d) Можно не сомневаться в том, что аксиома для действия по перемещению, составленная читателем, будет неправильной, поскольку, если объект слишком тяжел, то его позиция после применения оператора  $Push$  должна оставаться неизменной. Является ли это примером проблемы распространения последствий или проблемы спецификации? Исправьте составленное вами описание задачи, чтобы в нем учитывались тяжелые объекты.

**11.5.** Объясните, почему процесс формирования преемников в обратном поиске не требует добавления литералов, представляющих собой отрицательные результаты в рассматриваемом действии.

- 11.6.** Объясните, почему удаление отрицательных результатов из каждой схемы действия в задаче Strips приводит к получению ослабленной задачи.
- 11.7.** Изучите определение **дву направленного поиска** в главе 3.
- Была бы идея дву направленного поиска в пространстве состояний перспективной для использования в планирования?
  - А что можно сказать применительно к дву направленному поиску в пространстве планов с частичным упорядочением?
  - Разработайте версию планирования с частичным упорядочением, в которой любое действие может быть добавлено к плану, если выполнения его предусловий можно добиться с помощью результатов действий, уже имеющихся в плане. Объясните, как следует поступать с конфликтами и ограничениями упорядочения. Является ли этот алгоритм по сути идентичным прямому поиску в пространстве состояний?
  - Рассмотрите планировщик с частичным упорядочением, в котором сочетается метод, описанный в упр. 11.7, в, со стандартным методом добавления действий для достижения открытых условий. Будет ли результирующий алгоритм таким же, как описано в упр. 11.7, б?
- 11.8.** Сконструируйте уровни 0, 1 и 2 графа планирования для задачи, приведенной в листинге 11.1.
- 11.9.** Докажите приведенные ниже утверждения о графе планирования.
- Литерал, который не появляется на заключительном уровне графа, не может быть достигнут.
  - Уровневая стоимость литерала в последовательном графе не больше фактической стоимости оптимального плана его достижения.
- 11.10.** Авторы подчеркнули различие между планировщиками с прямым и обратным поиском в пространстве состояний и планировщиками с частичным упорядочением, указав, что последние представляют собой средства поиска в пространстве планов. Объясните, на основании чего средства прямого и обратного поиска в пространстве состояний можно также рассматривать как средства поиска в пространстве планов, и укажите, каковыми являются операторы уточнения плана.
- 11.11.** На рис. 11.8 показана задача в мире блоков, известная как **аномалия Зюссмана**. Эта задача рассматривается как аномальная, поскольку планировщики без чередования, применяющиеся в начале 1970-х годов, не могли ее решить. Запишите определение этой задачи в системе обозначений Strips и решите ее либо вручную, либо с использованием программы планирования. Планировщиком без чередования является такой планировщик, который после получения двух подцелей,  $G_1$  и  $G_2$ , вырабатывает либо план для  $G_1$ , который соединяется с планом  $G_2$ , либо наоборот. Объясните, почему планировщик без чередования не мог решить эту задачу.

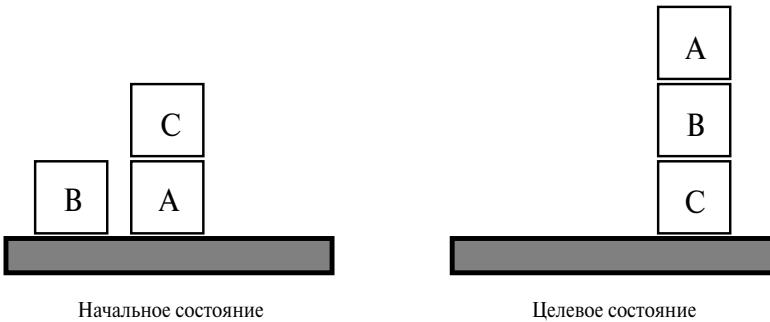


Рис. 11.8. Задача планирования в мире блоков, получившая название аномалии Зюссмана

**11.12.** Рассмотрите задачу надевания туфель и носков, которая определена в разделе 11.3. Примените для решения этой задачи алгоритм Graphplan и покажите полученное решение. Добавьте действия для надевания пальто и шляпы. Продемонстрируйте план с частичным упорядочением, который представляет собой одно из решений, и покажите, что существует 180 различных линеаризаций плана с частичным упорядочением. Каково минимальное количество различных решений с графом планирования, необходимых для представления всех 180 линеаризаций?

**11.13.** Первоначальная программа Strips была разработана для управления роботом Shakey. На рис. 11.9 показана версия мира Shakey, состоящего из четырех комнат (*Room*), расположенных вдоль коридора (*Corridor*), где в каждой комнате имеется дверь (*Door*) и выключатель света (*Switch*).

Действия в мире робота Shakey включают перемещение из одного места в другое, передвижение подвижных объектов (таких как ящики), подъем и спуск с прочных объектов (таких как ящики), а также включение и выключение выключателей света. Сам этот робот никогда не был достаточно находчивым, чтобы забраться на ящик или щелкнуть выключателем, но планировщик Strips оказался способным находить и выводить на печать планы, превосходящие мыслительные способности робота. Ниже перечислены шесть действий робота Shakey.

- Действие  $Go(x, y)$ , которое требует, чтобы Shakey был в позиции  $x$ , и определяет, что позиции  $x$  и  $y$  находятся в одной и той же комнате. В соответствии с общепринятым соглашением дверь между двумя комнатами считается находящейся одновременно в обеих этих комнатах.
- Действие по перемещению ящика  $b$  из позиции  $x$  в позицию  $y$  в пределах одной той же комнаты,  $Push(b, x, y)$ . Нам потребуются предикат *Box* и константы для описания ящиков.
- Действия по подъему на ящик,  $ClimbUp(b)$ , и спуску с ящика,  $ClimbDown(b)$ . Нам потребуются предикат *On* и константа *Floor* (Пол).
- Действия по включению выключателя света,  $TurnOn(s)$ , и его выключению,  $TurnOff(s)$ . Для того, чтобы включить или выключить свет, робот Shakey должен стоять на ящике, расположенном в том месте, где находится выключатель света.

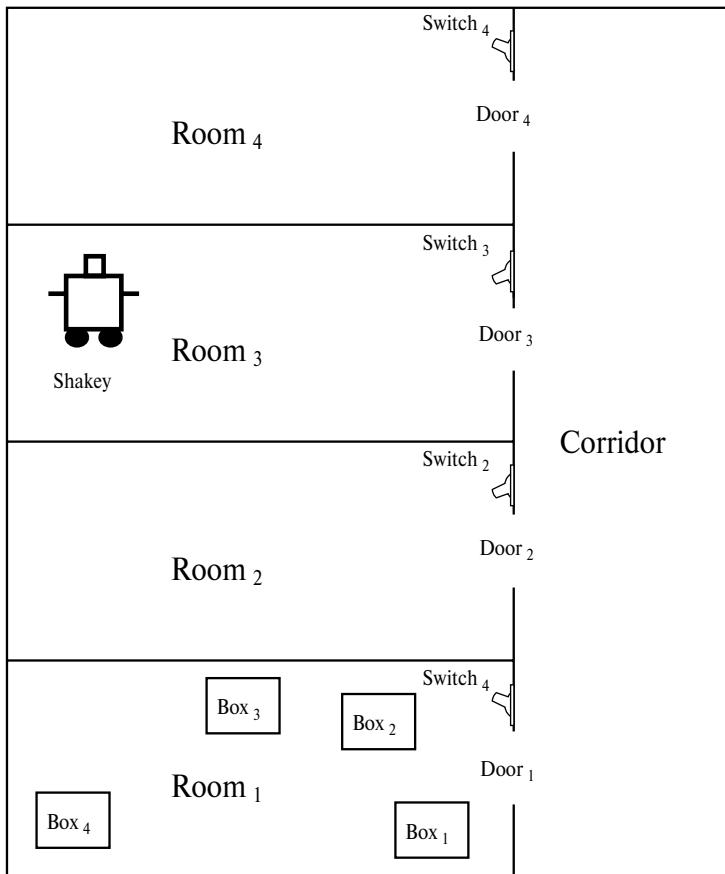


Рис. 11.9. Мир робота Shakey. Робот Shakey может передвигаться между отметками в пределах комнаты, проходить через двери между комнатами и коридором, взбираться на те объекты, на которые ему разрешено взбираться, и двигать те объекты, которые ему разрешено двигать, а также может щелкать выключателями света

Опишите в системе обозначений Strips шесть действий робота Shakey и показанное на рис. 11.9 начальное состояние. Составьте план, с помощью которого робот Shakey мог бы переместить ящик *Box<sub>2</sub>* в комнату *Room<sub>2</sub>*.

**11.14.** В этой главе встречались только такие графы планирования, которые позволяли использовать лишь пропозициональные действия. А что было бы, если возникла необходимость применять графы планирования для задач с переменными в цели, таких как  $At(P_1, x) \wedge At(P_2, x)$ , где  $x$  пробегает по конечной области определения местонахождений? Как можно было бы представить такую задачу, чтобы для ее решения могли использоваться графы планирования? (Подсказка. Вспомните действие *Finish* из области планирования POP. Какие предусловия оно должно иметь?)

**11.15.** Вплоть до настоящего момента предполагалось, что действия выполняются только в подходящих для этого ситуациях. Рассмотрим, что должны утвер-

ждать пропозициональные аксиомы состояния-преемника, такие как уравнение 11.1, в отношении действий, предусловия которых не выполнены.

- a) Покажите, что эти аксиомы предсказывают, что ничего не произойдет, если действие будет осуществлено в таком состоянии, в котором не выполнены его предусловия.
- б) Рассмотрите план  $p$ , который содержит действия, требуемые для достижения цели, но включает также недопустимые действия. Является ли истинным или ложным приведенное ниже высказывание?

*Начальное состояние  $\wedge$  Аксиомы состояния-преемника  $\wedge p \models \text{Цель}$*

- в) Возможно ли с помощью аксиом состояния-преемника первого порядка в ситуационном исчислении (как в главе 10) доказать, что план, содержащий недопустимые действия, позволяет достичь цели?

**11.16.** Приводя примеры из проблемной области грузового аэропорта, объясните, каким образом метод расщепления символов позволяет уменьшить размеры аксиом предусловия и аксиом исключения действий. Выведите общую формулу для оценки размера каждого множества аксиом в терминах количества временных этапов, количества схем действий, их арностей и количества объектов.

**11.17.** В алгоритме SATplan, приведенном в листинге 11.7, предусмотрено, что каждый вызов алгоритма проверки выполнимости подтверждает цель  $g^T$ , где  $T$  находится в пределах от 0 до  $T_{\max}$ . Предположим вместо этого, что алгоритм проверки выполнимости вызывается только один раз, с целью  $g^0 \vee g^1 \vee \dots \vee g^{T_{\max}}$ .

- а) Будет ли такой вызов всегда возвращать план с длиной, меньшей или равной  $T_{\max}$ , если таковой существует?
- б) Приведет ли такой подход к появлению каких-либо новых фиктивных “решений”?
- в) Обсудите, как можно было бы модифицировать алгоритм проверки выполнимости, такой как WalkSAT, чтобы он находил короткие решения (если они существуют) после получения дизъюнктивной цели в указанной выше форме.

# 12 ПЛАНИРОВАНИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ В РЕАЛЬНОМ МИРЕ

*В этой главе показано, что более выразительные представления и более интерактивные архитектуры агентов обеспечивают возможность создания планировщиков, применимых в реальном мире.*

В предыдущей главе описывались наиболее важные понятия, представления и алгоритмы для планирования. Планировщики, применяемые в реальном мире для решения таких задач, как планирование наблюдений с помощью космического телескопа Хаббл, управление заводами и фабриками, а также осуществление поставок для военных кампаний, являются более сложными; они превосходят свои более простые аналоги и с точки зрения языка представления, и с точки зрения того способа, который применяется в планировщике для взаимодействия с его средой. Такие различия описаны в настоящей главе. В разделе 12.1 рассматриваются задачи планирования и составления расписаний с учетом временных и ресурсных ограничений, а в разделе 12.2 описывается планирование с помощью предопределенных субпланов. В разделах 12.3–12.6 рассматривается ряд архитектур агентов, предназначенных для осуществления действий в неопределенных вариантах среды. В разделе 12.7 показано, как должны составляться планы, когда среда содержит и других агентов.

## 12.1. ВРЕМЯ, РАСПИСАНИЯ И РЕСУРСЫ

В представлении на языке Strips говорится о том, какие действия должны быть выполнены, но это представление основано на ситуационном исчислении, поэтому с его помощью нельзя показать, сколько времени потребуется на проведение того или иного действия, или даже обозначить время, когда должно произойти это действие, помимо указаний на то, что оно должно быть осуществлено до или после дру-

гого действия. Но применительно к некоторым проблемным областям было бы желательно также указывать, когда должны начинаться и оканчиваться действия. Например, в проблемной области транспортировки грузов может потребоваться знать, когда именно прибывает самолет, перевозящий некоторый груз, а не просто пользоваться информацией о том, что он прибудет, когда закончится полет.

Время является существенно важным фактором для крупного семейства приложений, называемых задачами **планирования производства**. Для выполнения таких задач требуется осуществление ряда работ, каждая из которых состоит из последовательности действий, а каждое действие имеет определенную продолжительность и может потребовать определенных ресурсов. Проблема состоит в разработке расписания, которое минимизирует общее время, требуемое для выполнения всех работ, при соблюдении ограничений на ресурсы.

Пример задачи планирования производства приведен в листинге 12.1. Это — в высшей степени упрощенная задача сборки автомобиля. В ней представлены две работы: сборка автомобилей  $C_1$  и  $C_2$ . Каждая работа состоит из трех действий: установка двигателя, установка колес и проверка результатов. Двигатель должен устанавливаться в первую очередь (поскольку в автомобиле этой модели с установленными передними колесами затрудняется доступ к двигательному отсеку), а проверка, безусловно, должна проводиться в последнюю очередь.

**Листинг 12.1.** Задача планирования производства, связанная со сборкой двух автомобилей. Обозначение  $\text{Duration}(d)$  показывает, что для выполнения некоторого действия требуется  $d$  минут, а обозначение  $\text{Engine}(E_1, C_1, 60)$  показывает, что  $E_1$  — это двигатель, который устанавливается в шасси  $C_1$ , и для его установки требуется 60 минут

---

```

Init(Chassis(C1) ∧ Chassis(C2)
     ∧ Engine(E1, C1, 30) ∧ Engine(E2, C2, 60)
     ∧ Wheels(W1, C1, 30) ∧ Wheels(W2, C2, 15))
Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e, c)
       Precond: Engine(e, c, d) ∧ Chassis(c) ∧ ¬EngineIn(c),
       Effect: EngineIn(c) ∧ Duration(d))
Action(AddWheels(w, c),
       Precond: EngineIn(c) ∧ Wheels(w, c, d) ∧ Chassis(c),
       Effect: WheelsOn(c) ∧ Duration(d))
Action(Inspect(c),
       Precond: EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c),
       Effect: Done(c) ∧ Duration(10))

```

---

Задача, приведенная в листинге 12.1, может быть решена с помощью любого из планировщиков, которые уже описывались в настоящей книге. На рис. 12.1 показано решение, которое было бы получено с помощью планировщика POP с частичным упорядочением (если игнорируются некоторые числовые данные). Теперь для преобразования этой задачи в задачу составления расписания, а не в задачу планирования, необходимо определить, когда должно начаться и закончиться каждое действие, с учетом продолжительности действия, а также их упорядочения. Обозначение  $\text{Duration}(d)$  в спецификации результата действия (где переменная  $d$  должна быть привязана к некоторому числу) показывает, что для выполнения действия требуется  $d$  минут.

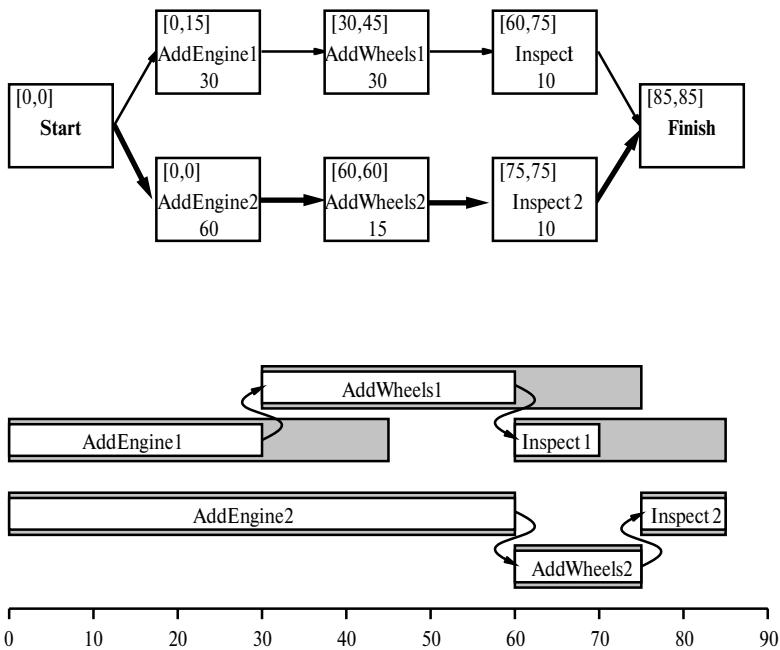


Рис. 12.1. Решение задачи планирования производства, приведенной в листинге 12.1. В верхней части этого рисунка решение показано в виде плана с частичным упорядочением. Продолжительность каждого действия обозначена в нижней части каждого прямоугольника, а значения самого раннего и самого позднего времени начала, условно обозначаемые как  $[ES, LS]$ , показаны в верхней левой части. Разность между этими двумя числами представляет собой резерв времени для действия; действия с нулевым резервом находятся на критическом пути, который обозначен жирными стрелками. В нижней части рисунка то же решение приведено в виде временного графика. Серые прямоугольники показывают интервалы времени, в течение которых может быть выполнено некоторое действие, при условии, что соблюдаются ограничения упорядочения. Незанятая часть серого прямоугольника обозначает резерв времени

Определив частичное упорядочение действий с учетом их продолжительности, как показано на рис. 12.1, можно применить **метод критического пути** (Critical Path Method — CPM) для выяснения допустимых значений времени начала и окончания каждого действия. Путем через план с частичным упорядочением называется линейно упорядоченная последовательность действий, начинающаяся в состоянии *Start* и оканчивающаяся в состоянии *Finish* (например, в плане с частичным упорядочением, показанном на рис. 12.1, есть два пути).

**Критическим** называется путь, имеющий максимальную суммарную продолжительность; этот путь называется “критическим”, поскольку от него зависит продолжительность выполнения всего плана, — сокращение продолжительности других путей не приведет к сокращению времени выполнения всего плана в целом, но задержка начала любого действия в критическом пути замедлит осуществление всего плана. На этом рисунке критический путь показан жирными линиями. Для осуществления всего плана за минимальное суммарное время действия в критическом пу-

ти следует выполнять без задержки между ними. С другой стороны, действия, лежащие вне критического пути, имеют определенный запас времени — временное окно, в течение которого они могут быть выполнены. Это окно задается в терминах самого раннего возможного времени начала,  $ES$ , и самого позднего возможного времени начала,  $LS$ . Разность  $LS - ES$  называется **резервом времени** для действия. Как показано на рис. 12.1, для выполнения всего плана потребуется 85 минут, каждое действие в критическом пути имеет резерв времени 0 (такое условие имеет место во всех расписаниях), а каждое действие в работе по сборке  $C_1$  имеет десятиминутное окно, в пределах которого оно может быть начато. Значения времени  $ES$  и  $LS$  для всех действий, вместе взятые, составляют **расписание**, представляющее собой решение данной задачи.

Ниже приведены формулы, которые служат определением значений  $ES$  и  $LS$ , а также лежат в основе алгоритма динамического программирования, применяемого для их вычисления.

$$\begin{aligned} ES(Start) &= 0 \\ ES(B) &= \max_{A \in B} ES(A) + Duration(A) \\ LS(Finish) &= ES(Finish) \\ LS(A) &= \min_{B \in A} LS(B) - Duration(A) \end{aligned}$$

Идея этого алгоритма состоит в том, что вначале следует присвоить терму  $ES(Start)$  значение 0. Затем, как только будет выявлено действие  $B$ , такое, что все действия, непосредственно предшествующие  $B$ , имеют присвоенные значения  $ES$ , терму  $ES(B)$  присваивается максимальное из самых ранних значений времени завершения этих непосредственно предшествующих действий, где самое раннее время завершения действия определяется как самое раннее время начала плюс его продолжительность. Этот процесс повторяется до тех пор, пока каждому действию не присваивается значение  $ES$ . Значения  $LS$  вычисляются аналогичным образом, в ходе передвижения в обратном направлении от действия  $Finish$ . Проработку деталей этого алгоритма оставляем читателю в качестве упражнения.

Сложность алгоритма критического пути составляет всего лишь  $O(Nb)$ , где  $N$  — количество действий;  $b$  — максимальный коэффициент ветвления на входе или выходе любого действия. (Чтобы убедиться в этом, достаточно отметить, что вычисления значений  $LS$  и  $ES$  осуществляются по одному разу для каждого действия, а в каждом вычислении происходит итерация, самое большое, по  $b$  других действий.) Поэтому, если дано частичное упорядочение действий, задача поиска расписания с минимальной продолжительностью решается очень просто.

### Составление расписаний с ресурсными ограничениями

Реальные задачи составления расписаний усложняются из-за наличия ограничений на **ресурсы**. Например, для установки в автомобиль двигателя требуется лебедка для двигателя. Если есть только одна лебедка, то нельзя одновременно устанавливать двигатель  $E_1$  в автомобиль  $C_1$  и двигатель  $E_2$  в автомобиль  $C_2$ , поэтому расписание, показанное на рис. 12.1, будет неосуществимым. В данном примере лебедка для двигателя представляет собой пример **повторно применяемого ресурса** — ресурса, который “занят” во время действия, но снова становится доступным

после завершения этого действия. Следует отметить, что повторно применяемые ресурсы невозможно учесть в нашем стандартном описании действий в терминах предусловий и результатов, поскольку количество доступных ресурсов после завершения действия остается неизменным<sup>1</sup>. По этой причине дополним наше представление, включив в него поле в форме `Resource: R(k)`, которое означает, что для выполнения данного действия требуются  $k$  единиц ресурса  $R$ . Требования к ресурсам являются одновременно и предпосылкой (действие не может быть выполнено, если ресурс недоступен), и временным результатом, в том смысле, что во время выполнения действия доступность ресурса  $x$  сокращается на  $k$  единиц. В листинге 12.2 показано, как дополнить задачу сборки автомобилей для включения трех ресурсов — лебедки для двигателя, с помощью которой устанавливаются двигатели, станции монтажа колес, на которой устанавливаются колеса, и двух контролёров. На рис. 12.2 показано решение с самым быстрым временем завершения, 115 минут. Это время больше по сравнению с 80 минутами, которые требовались для выполнения расписания без ресурсных ограничений. Следует отметить, что нет такого промежутка времени, в который требовались бы оба контролёра, поэтому, составив данное расписание, можно сразу же перевести одного из двух контролёров на другой участок, где он будет приносить больше пользы.



*Рис. 12.2. Решение задачи составления производственного расписания с ресурсами, приведенной в листинге 12.2. На левом поле перечислены три ресурса, а действия показаны с выравниванием по горизонтали с теми ресурсами, которые в них используются. Существуют два возможных расписания, зависящих от того, при сборке какого автомобиля первой используется станция установки двигателя; здесь показано оптимальное решение, которое занимает 115 минут*

**Листинг 12.2.** Задача составления производственного расписания для сборки двух автомобилей с учетом ресурсов. Доступными ресурсами являются одна станция сборки двигателя, одна станция сборки колес и два контролёра. Обозначение `Resource: x` указывает, что ресурс  $x$  используется во время выполнения действия, но снова становится свободным после завершения этого действия

```

Init(Chassis(C1) ∧ Chassis(C2)
     ∧ Engine(E1, C1, 30) ∧ Engine(E2, C2, 60)
     ∧ Wheels(W1, C1, 30) ∧ Wheels(W2, C2, 15)
     ∧ EngineHoists(1) ∧ WheelStations(1) ∧ Inspectors(2))
Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e, c),
       Precond: Engine(e, c, d) ∧ Chassis(c) ∧ ¬EngineIn(c),

```

<sup>1</sup> В отличие от этого, **потребляемые ресурсы**, такие как винты для монтажа двигателя, могут учитываться в рамках стандартной инфраструктуры; см. упр. 12.2.

```

Effect: EngineIn(c) ∧ Duration(d),
Resource: EngineHoists(1)
Action(AddWheels(w, c),
Precond: EngineIn(c) ∧ Wheels(w, c, d) ∧ Chassis(c),
Effect: WheelsOn(c) ∧ Duration(d),
Resource: WheelStations(1))
Action(Inspect(c),
Precond: EngineIn(c) ∧ WheelsOn(c),
Effect: Done(c) ∧ Duration(10),
Resource: Inspectors(1))

```

---

Представление ресурсов в виде числовых количественных значений, таких как *Inspectors*(2), а не именованных сущностей, таких как *Inspector*( $I_1$ ) и *Inspector*( $I_2$ ), может служить примером очень продуктивного метода, называемого **агрегированием**. Основная идея агрегирования состоит в том, что отдельные объекты должны группироваться в количественные величины, если все объекты неразличимы применительно к рассматриваемому назначению. В данной задаче сборки не имеет значения, какой именно контролёр проверит автомобиль, поэтому нет необходимости проводить между ними различие. (Та же идея может применяться при решении задачи с миссионерами и каннибалами, приведенной в упр. 3.9.) Агрегирование представляет собой важный способ уменьшения сложности задач. Рассмотрим, что произойдет, если будет предложено расписание, в котором имеется 10 одновременных действий по проверке *Inspect*, но в наличии есть только 9 контролёров. Если контролёры представлены с помощью количественных величин, неудача будет обнаружена немедленно и алгоритм выполнит возврат, чтобы попытаться составить другое расписание. А если бы контролёры были представлены как отдельные лица, то в алгоритме пришлось бы выполнять возвраты для опробования всех  $10!$  способов назначения контролёров для выполнения действий *Inspect*, притом что положительный результат так и не был бы получен.

Несмотря на их преимущества, ресурсные ограничения приводят к усложнению задач планирования, поскольку в них вводятся дополнительные зависимости между действиями. К тому же составление расписания без ограничений с использованием метода критического пути является несложным, а задача поиска расписания с ресурсными ограничениями, характеризующегося самым ранним возможным временем завершения, является NP-трудной. Такая сложность часто наблюдается не только в теории, но и на практике. Одно из заданий, которое предложили исследователям для проверки их сил в 1963 году (найти оптимальное расписание для задачи, в которой рассматриваются только 10 машин и 10 работ по 100 действий каждая), не удавалось решить в течение 23 лет [900]. Для его решения было опробовано много подходов, включая метод ветвей и границ, алгоритм эмуляции отжига, поиск с запретами, метод удовлетворения ограничений и другие методы, описанные в части II этой книги. Одной из простых, но широко применяемых эвристик является алгоритм с **минимальным резервом**. В нем планирование действий осуществляется в режиме жадного поиска. Во время каждой итерации в этом алгоритме рассматриваются не введенные в расписание действия, для которых в расписании заданы все их преемники, и в расписание вводится то действие, которое имеет наименьший резерв для самого раннего возможного времени начала. После этого в алгоритме обновляются значения времени *ES* и *LS* для каждого затронутого действия и повторя-

ется итерация. Эвристика основана на том же принципе, что и эвристика с “наиболее ограниченной переменной” в задачах удовлетворения ограничений. Этот алгоритм хорошо работает на практике, но применительно к рассматриваемой задаче сборки он привел к получению 130-минутного решения, а не 115-минутного решения, показанного на рис. 12.2.

Подход, принятый в этом разделе, состоит в том, что “вначале следует оставлять план, а затем расписание”. Это означает, что общая задача подразделяется на этап планирования, в котором выбираются и частично упорядочиваются действия для достижения целей данной задачи, и на следующий за ним этап составления расписания, в котором в план вводится временная информация для обеспечения того, чтобы он соответствовал ограничениям по ресурсам и срокам. Такой подход широко применяется в организациях, занимающихся реальным планированием производства и поставок, в которых этап планирования часто выполняется экспертами-людьми. Однако, если существуют жесткие ограничения на ресурсы, могут возникать такие ситуации, что одни допустимые планы приводят к получению гораздо более лучших графиков, чем другие. В этом случае имеет смысл интегрировать этапы планирования и составления расписаний, принимая во внимание продолжительности и совпадения действий во времени на этапе создания плана с частичным упорядочением. Для учета такой информации могут быть дополнены некоторые алгоритмы планирования, описанные в главе 11. Например, планировщики с частичным упорядочением могут обнаруживать нарушения ресурсных ограничений во многом с помощью такого же способа, который в них используется для обнаружения конфликтов с помощью причинных связей. А эвристики могут быть модифицированы для оценки общего времени завершения плана, а не просто суммарной стоимости всей действий. В настоящее время в этой области исследований ведется активная работа.

## 12.2. ПЛАНИРОВАНИЕ ИЕРАРХИЧЕСКОЙ СЕТИ ЗАДАЧ

Одной из наиболее привлекательных идей в области решения сложных задач является **иерархическая декомпозиция**. На основе иерархии процедур или классов объектов создается сложное программное обеспечение; из иерархии боевых подразделений складываются армии; иерархии отделов, филиалов и дочерних организаций лежат в основе правительств и корпораций. Основное преимущество иерархической структуры состоит в том, что на каждом уровне иерархии вычислительная задача, военная операция или административная функция сводится к небольшому количеству действий, выполняемых на более низком уровне, поэтому вычислительная стоимость поиска правильного способа упорядочения этих действий для решения текущей задачи очень невелика. С другой стороны, в неиерархических методах задача сводится к очень большому количеству отдельных действий; при решении крупномасштабных задач такой подход становится полностью неприменимым. Но в наилучшем случае (в котором высокоуровневые решения всегда сводятся к решениям, имеющим удовлетворительные низкоуровневые реализации) иерархические методы могут привести к созданию алгоритмов планирования с линейными, а не экспоненциальными затратами времени.

В настоящем разделе рассматривается метод планирования, основанный на **иерархических сетях задач**, или сетях HTN (Hierarchical Task Network). В приня-

том нами подхоле объединены идеи из области планирования с частичным упорядочением (раздел 11.3) и из области, известной под названием “планирование в иерархических сетях задач”, или планирование HTN. В планировании HTN первоначальный план, который описывает задачу, рассматривается как описание на очень высоком уровне того, что должно быть сделано, например строительство дома. Планы уточняются путем применения **декомпозиций действий**. В каждой декомпозиции действия одно действие высокого уровня сводится к частично упорядоченному множеству действий низкого уровня. Поэтому в декомпозициях действий воплощены знания о том, как осуществляются действия. Например, строительство дома может быть сведено к получению разрешения, найму подрядчика, осуществлению строительства и оплате работы подрядчика (такая декомпозиция показана на рис. 12.3). Этот процесс продолжается до тех пор, пока в плане не остаются только **примитивные действия**. Как правило, примитивными считаются такие действия, которые могут быть выполнены агентом автоматически. Например, для генерального подрядчика такое действие, как “оформление ландшафтной архитектуры”, может оказаться примитивным, поскольку оно сводится к привлечению подрядчика по ландшафтной архитектуре, а для подрядчика по ландшафтной архитектуре могут рассматриваться как примитивные действия, подобные “посадке на этом участке рододендронов”.

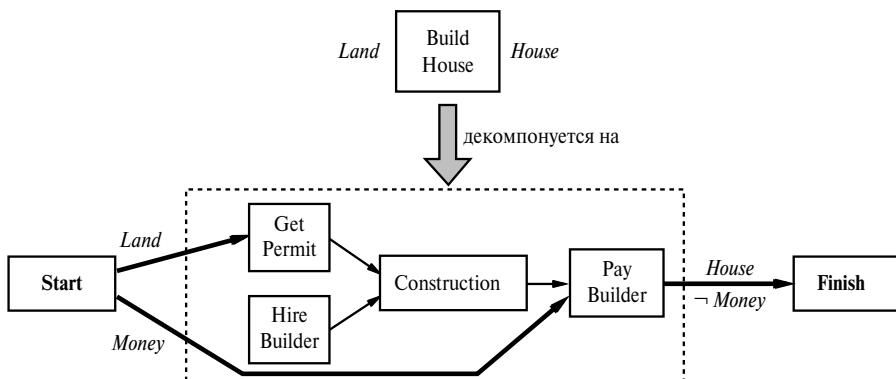


Рис. 12.3. Одна из возможных декомпозиций для действия BuildHouse

В “чистом” планировании HTN планы разрабатываются только путем последовательной декомпозиции действий. Поэтому планирование HTN может рассматриваться как процесс конкретизации описания некоторой деятельности, а не как процесс создания описания деятельности, начиная с пустого действия (как в случае планирования в пространстве состояний и планирования с частичным упорядочением). Как оказалось, каждое описание действия Strips может быть преобразовано в декомпозицию действия (см. упр. 12.6), а планирование с частичным упорядочением может рассматриваться как частный случай чистого планирования HTN. Однако для некоторых задач (особенно для тех, в которых применяется так называемая “новаторская” постановка с конъюнктивными целями) подход с использованием чистого планирования HTN становится не совсем естественным. Поэтому авторы предпочитают применять гибридный подход, в котором декомпозиции действий используются как уточнения плана в планировании с частичным упорядочением, в до-

полнение к стандартным операциям определения открытых условий и разрешения конфликтов путем введения ограничений упорядочения. (Подход, в котором планирование HTN рассматривается как расширение планирования с частичным упорядочением, имеет дополнительное преимущество в том, что могут использоваться те же соглашения по системе именования вместо введения полностью нового набора обозначений.) Начнем с более подробного описания декомпозиций действий. Затем рассмотрим, как можно модифицировать алгоритм планирования с частичным упорядочением для учета декомпозиций, и, наконец, рассмотрим вопросы полноты, сложности и практической применимости алгоритмов.

### Представление декомпозиций действий

Общие описания методов декомпозиции действий хранятся в **библиотеке планов**, из которой они извлекаются и конкретизируются для обеспечения потребностей формирования текущего плана. Каждый метод представляет собой выражение в форме *Decompose(a, d)*. Это выражение указывает, что может быть выполнена декомпозиция действия *a* на план *d*, представленный в виде плана с частичным упорядочением, как описано в разделе 11.3.

Строительство дома — это прекрасный, конкретный пример, поэтому мы будем использовать его для иллюстрации концепций декомпозиции действия. На рис. 12.3 показана одна возможная декомпозиция действия *BuildHouse* на четыре действия низкого уровня, а в листинге 12.3 приведены некоторые из описаний действий для данной проблемной области, а также декомпозиция действия *BuildHouse* в том виде, в каком она могла бы присутствовать в библиотеке планов. В этой библиотеке могут находиться также другие возможные декомпозиции.

**Листинг 12.3. Декомпозиции действий для задачи построения дома и подробная декомпозиция для действия *BuildHouse*.** В этих декомпозициях приняты упрощенное представление о деньгах, а также оптимистическое представление строителей в отношении перспектив оплаты

---

```

Action(BuyLand, Precond: Money, Effect: Land ∧ ¬Money)
Action(GetLoan, Precond: GoodCredit, Effect: Money ∧ Mortgage)
Action(BuildHouse, Precond: Land, Effect: House)

Action(GetPermit, Precond: Land, Effect: Permit)
Action(HireBuilder, Effect: Contract)
Action(Construction, Precond: Permit ∧ Contract,
      Effect: HouseBuilt ∧ ¬Permit)
Action(PayBuilder, Precond: Money ∧ HouseBuilt,
      Effect: ¬Money ∧ House ∧ ¬Contract)

Decompose(BuildHouse,
          Plan(Steps: {S1: GetPermit, S2: HireBuilder,
                      S3: Construction, S4: PayBuilder}
               Orderings: {Start < S1 < S3 < S4 < Finish, Start < S2 < S3},
               Links: {Start  $\xrightarrow{\text{Land}}$  S1, Start  $\xrightarrow{\text{Money}}$  S4,
                        S1  $\xrightarrow{\text{Permit}}$  S3, S2  $\xrightarrow{\text{Contract}}$  S3, S3  $\xrightarrow{\text{HouseBuilt}}$  S4,
                        S4  $\xrightarrow{\text{House}}$  Finish, S4  $\xrightarrow{\text{Money}}$  Finish}) )

```

---

В действии *Start* этой декомпозиции должны быть предусмотрены все те предусловия действий в плане, которые не предусмотрены в других действиях. Такие предусловия принято называть **внешними предусловиями**. В данном примере внешними предусловиями декомпозиции являются *Land* (Земельный участок) и *Money* (Деньги). Аналогичным образом, все **внешние результаты**, являющиеся предусловиями действия *Finish*, представляют собой такие результаты действий в плане, которые не отрицаются другими действиями. В рассматриваемом примере внешними результатами действия *BuildHouse* являются *House* (Дом) и *-Money* (Отсутствие денег). В некоторых планировщиках HTN проводится также различие между **первичными результатами**, такими как *House*, и **вторичными результатами**, такими как *-Money*. Для достижения целей могут использоваться только первичные результаты, тогда как оба рода результатов могут вызывать конфликты с другими действиями; они позволяют в значительной степени сократить пространство поиска<sup>2</sup>.

Декомпозиция должна представлять собой правильный способ осуществления действия. План *d* является правильным способом осуществления действия, если *d* — полный и согласованный план с частичным упорядочением для задачи достижения результатов действия *a* при наличии предусловий действия *a*. Очевидно, что декомпозиция будет правильной, если она стала результатом применения непротиворечивого планировщика с частичным упорядочением.

Библиотека планов может содержать несколько декомпозиций для любого конкретного действия высокого уровня; например, может существовать еще одна декомпозиция для действия *BuildHouse*, которая описывает такой процесс, что агент возводит дом-самостройку из камней и торфа голыми руками. Каждая декомпозиция должна представлять собой правильный план, но может иметь дополнительные предусловия и результаты, кроме тех, что указаны в описании действия высокого уровня. Например, декомпозиция для действия *BuildHouse*, показанная на рис. 12.3, кроме земельного участка (*Land*), требует денег (*Money*), и имеет такой результат, как *-Money*. С другой стороны, вариант с самостройкой не требует денег, но требует большого запаса камней ( *Rocks*) и торфа (*Turf*), а также может привести к результату *BadBack* (Больная спина).

Учитывая то, что действия высокого уровня, такие как *BuildHouse*, могут иметь несколько возможных декомпозиций, нельзя избежать такой ситуации, что их описания действий *Strips* будут скрывать некоторые из предусловий и результатов декомпозиций этих действий. Предусловия действия высокого уровня должны представлять собой пересечение внешних предусловий возможных декомпозиций этого действия, а результаты — пересечение внешних результатов его декомпозиций. Иначе говоря, необходимо обеспечить, чтобы предусловия и результаты высокого уровня представляли собой подмножества истинных предусловий и результатов каждой примитивной реализации действия высокого уровня.

Следует также отметить две другие формы сокрытия информации. Во-первых, в описании высокого уровня полностью игнорируются все **внутренние результаты** декомпозиций. Например, в декомпозиции действия *BuildHouse* имеются времен-

<sup>2</sup> Это позволяет также предотвратить обнаружение неочевидных планов. Например, некоторое лицо, которому грозит процедура банкротства, может избавиться от всех ликвидных активов (т.е. достичь цели *-Money*), купив или построив дом. Этот план полезен, поскольку современные законы запрещают отчуждать в пользу кредиторов основное жилье.

ные внутренние результаты *Permit* (Наличие разрешения) и *Contract* (Заключение контракта)<sup>3</sup>. Во-вторых, в описании высокого уровня не указаны интервалы “внутри” действия, в течение которых должны иметь место предусловия и результаты высокого уровня. Например, предусловие *Land* должно оставаться истинным (в нашей очень упрощенной модели) только до выполнения действия *GetPermit* (Получение разрешения), а результат *House* становится истинным только после выполнения действия *PayBuilder* (Оплата работы подрядчика).

Скрытие информации такого рода становится очень важным, если иерархическое планирование применяется для уменьшения сложности; мы должны иметь возможность рассуждать о действиях высокого уровня, не заботясь о бесчисленных подробностях реализации. Однако за такую возможность приходится платить. Например, могут существовать конфликты между внутренними условиями одного действия высокого уровня и внутренними действиями другого, но способа обнаружить эту ситуацию с помощью высокоуровневых описаний не существует. Такая проблема оказала значительно влияние на алгоритмы планирования HTN. По сути, примитивные действия могут рассматриваться в алгоритме планирования как точечные события, а действия высокого уровня имеют временную протяженность, в пределах которой могут происходить другие всевозможные события.

### Модификация планировщика для его использования в сочетании с декомпозициями

В этом разделе описано, как модифицировать алгоритм POP для его применения в планировании HTN. Для этого мы модифицируем функцию определения преемника POP (с. 530), чтобы иметь возможность применять методы декомпозиции к текущему частичному плану *P*. Новые планы определения преемника формируются так, что вначале выбирается некоторое непримитивное действие *a'* в плане *P*, а затем для любого метода *Decompose(a, d)* из библиотеки планов, такого, что *a* и *a'* унифицируются с помощью подстановки  $\theta$ , действие *a'* заменяется декомпозицией *d' = Subst(\theta, d)*.

Один из примеров применения такого метода показан на рис. 12.4. В верхней части приведен план *P* возведения дома. Для декомпозиции выбирается действие высокого уровня *a' = BuildHouse*. В качестве декомпозиции *d* берется план, приведенный на рис. 12.3, и действие *BuildHouse* заменяется этой декомпозицией. Затем вводится дополнительный этап *GetLoan* (Получение ссуды) для разрешения нового открытого условия *Money*, которое создается на данном этапе декомпозиции. Замена действия его декомпозицией немного напоминает пересадку органов в хирургии: мы должны вынуть новый субплан из его упаковки (этапов *Start* и *Finish*), вставить его в нужное место и правильно связать сосуды, ткани и нервы. Для решения такой задачи может применяться несколько методов. Точнее, для каждой возможной декомпозиции *d'* должны быть выполнены описанные ниже этапы.

---

<sup>3</sup> Действие *Construction* отрицает результат *Permit*, поскольку в противном случае одно и то же разрешение можно было бы использовать для строительства нескольких домов. Но, к сожалению, действие *Construction* не завершает действие *Contract*, поскольку вначале необходимо расстаться с деньгами и совершить действие *PayBuilder*.

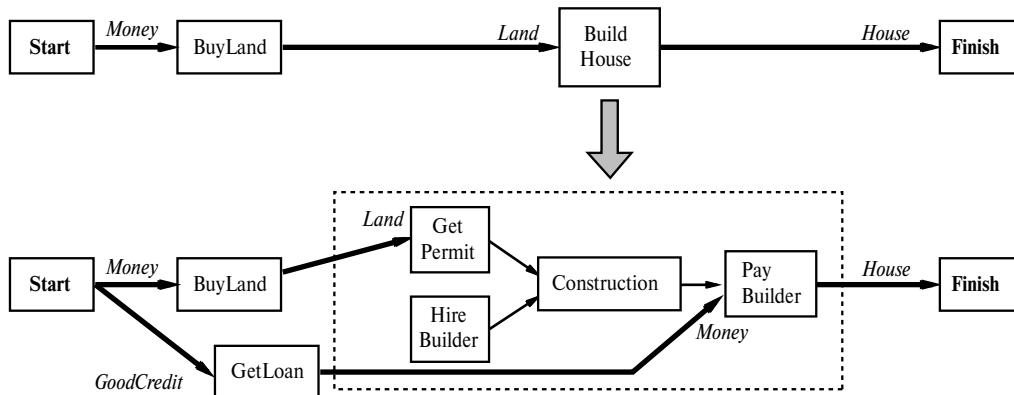


Рис. 12.4. Декомпозиция действия высокого уровня в существующем плане. Действие *BuildHouse* заменяется его декомпозицией, взятой из плана, показанного на рис. 12.3. Внешнее предусловие *Land* выполняется с помощью существующей причинной связи от действия *BuyLand* (*Покупка земельного участка*). После выполнения этого этапа декомпозиции внешнее предусловие *Money* остается открытым, поэтому вводится новое действие, *GetLoan* (*Получение ссуды*)

1. Вначале действие  $a'$  удаляется из плана  $P$ . Затем для каждого этапа  $s$  в декомпозиции  $d'$  необходимо выбрать действие для выполнения соответствующей роли в  $s$  и добавить его к плану. Это может быть либо новый экземпляр этапа  $s$ , либо существующий этап  $s'$  из плана  $P$ , который унифицируется с этапом  $s$ . Например, декомпозиция действия *MakeWine* (*Выращивание винограда*) может потребовать, чтобы мы купили земельный участок, *BuyLand*; но, по-видимому, достаточно будет использовать то же действие *BuyLand*, которое уже было предусмотрено в плане строительства дома. Такая ситуация будет называться **совместным использованием подзадач**.

На рис. 12.4 возможности совместного использования подзадач отсутствуют, поэтому создаются новые экземпляры действий. После выбора действий в них копируются все внутренние ограничения из декомпозиции  $d'$ , например, такие, что действие *GetPermit* должно быть выполнено перед действием *Construction* и что есть причинная связь между этими двумя этапами, обуславливающая выполнение предусловия *Permit* действия *Construction*. На этом задача замены  $a'$  конкретизацией декомпозиции  $d$  на основе подстановки  $\theta$  завершается.

2. На следующем этапе необходимо связать ограничения упорядочения для  $a'$  в первоначальном плане с этапами декомпозиции  $d'$ . Вначале рассмотрим любое ограничение упорядочения в плане  $P$ , которое задано в форме  $B \prec a'$ . Как должно быть упорядочено действие  $B$  по отношению к этапам декомпозиции  $d'$ ? Наиболее очевидное решение состоит в том, что действие  $B$  должно быть завершено перед выполнением любого этапа декомпозиции  $d'$ , а этого можно добиться, заменяя каждое ограничение в форме  $Start \prec s$  в декомпозиции  $d'$  ограничением  $B \prec s$ . С другой стороны, этот подход может оказаться слишком жестким! Например, действие *BuyLand* должно быть выполнено перед действием *BuildHouse*, но нет безусловной необходимости

сти, чтобы в расширенном плане покупка земли *BuyLand* осуществлялась перед наймом подрядчика *HireBuilder*. Наложив слишком строгое упорядочение, мы можем исключить возможность обнаружения некоторых решений. Поэтому лучший подход состоит в том, чтобы в каждом ограничении упорядочения записывались причины введения этого ограничения; в таком случае при развертывании действия высокого уровня можно будет применять настолько более ослабленные новые ограничения упорядочения, насколько это возможно, в полном соответствии с причиной введения первоначального ограничения. Точно такие же соображения могут применяться при замене ограничений в форме  $a' \prec C$ .

3. Конечный этап состоит в увязке причинных связей. Если одной из причинных связей в первоначальном плане была  $B \xrightarrow{p} a'$ , она заменяется множеством причинных связей от  $B$  ко всем этапам декомпозиции  $d'$  с предусловиями  $p$ , которые выполнены на этапе *Start* декомпозиции  $d$  (т.е. ко всем этапам декомпозиции  $d'$ , для которых  $p$  является внешним предусловием). В данном примере причинная связь  $BuyLand \xrightarrow{\text{Land}} BuildHouse$  заменяется связью  $BuyLand \xrightarrow{\text{Land}} Permit$ . (Предусловие *Money* для действия *PayBuilder* в этой декомпозиции становится открытым условием, поскольку ни в одном из действий в первоначальном плане не выполнено действие по получению денег *Money* для строительства дома *BuildHouse*.) Аналогичным образом, для каждой причинной связи  $a' \xrightarrow{p} C$  в плане необходимо предусмотреть ее замену множеством причинных связей к  $C$  от каждого этапа декомпозиции  $d'$ , в котором выполняется предусловие  $p$  для этапа *Finish* в декомпозиции  $d$  (т.е. от этапа в декомпозиции  $d'$ , для которого предусловие  $p$  является внешним результатом). В данном примере связь  $BuildHouse \xrightarrow{\text{House}} Finish$  заменяется связью  $PayBuilder \xrightarrow{\text{House}} Finish$ .

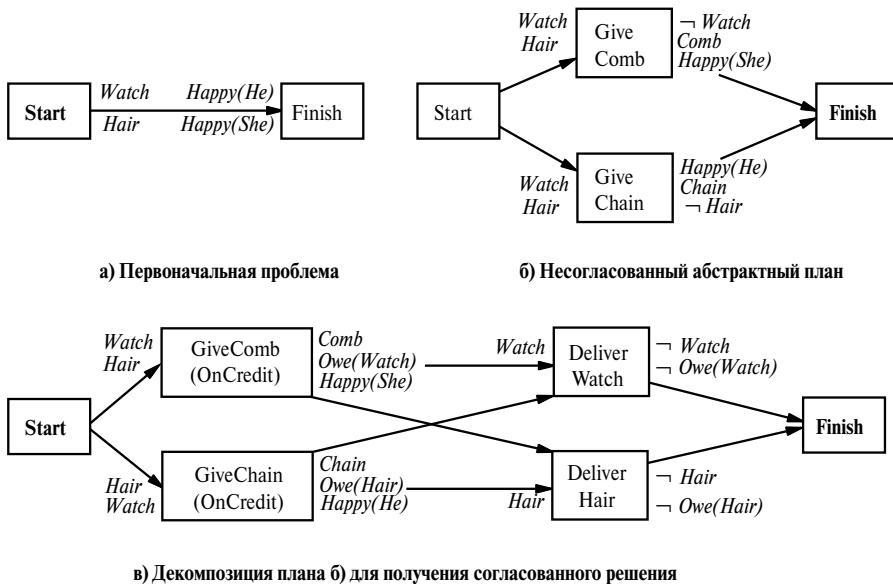
На этом завершаются<sup>4</sup> дополнения, требуемые для формирования декомпозиций в контексте применения планировщика POP.

Необходимость в дополнительных модификациях алгоритма POP связана с тем, что действия высокого уровня скрывают информацию об их конечных примитивных реализациях. В частности, в первоначальном алгоритме POP осуществляется возврат с индикатором неудачи, если текущий план включает неразрешимый конфликт, т.е. если одно из действий конфликтует с причинной связью, но не может быть переупорядочено так, чтобы оно находилось до или после этой связи (подобный пример приведен на рис. 11.4). При использовании действий высокого уровня, с другой стороны, с виду неразрешимые конфликты иногда могут быть разрешены путем декомпозиции конфликтующих действий и чередования их этапов. Соответствующий пример приведен на рис. 12.5. Таким образом, может возникнуть такая ситуация, что путем декомпозиции может быть получен полный и согласованный план из примитивных действий, даже если не существует полного и согласованного плана из действий высокого уровня. Такая возможность означает, что полный планировщик HTN не должен использовать многие варианты отсечения, которые предусмотрены

---

<sup>4</sup> Должны быть также предусмотрены некоторые другие небольшие модификации, которые требуются для разрешения конфликтов в действиях высокого уровня; заинтересованный читатель может ознакомиться со статьями, указанными в конце данной главы.

для стандартного планировщика POP. Еще один способ организации работы может состоять в том, чтобы отсечение применялось в любом случае, в надежде на то, что ни одно решение не будет упущенено.



*Рис. 12.5. Задача “Дары волхвов”, взятая из одноименного рассказа О’Генри, может служить примером несогласованного абстрактного плана, который, тем не менее, может быть декомпонован в согласованное решение. На рис. 12.5, а приведена постановка задачи: у пары бедняков есть только две ценные вещи — у него золотые часы, а у нее прекрасные длинные волосы. Каждый из супругов собирается купить подарок, чтобы сделать другому приятное. Он собирается продать свои часы, чтобы купить серебряный гребень для ее волос, а она собирается продать свои волосы, чтобы купить золотую цепочку для его часов. Частичный план, показанный на рис. 12.5, б, является несогласованным, поскольку не существует способа упорядочить абстрактные этапы GiveComb (Подарить гребень) и GiveChain (Подарить цепочку) без конфликта. (Мы предполагаем, что действие GiveComb имеет предусловие Hair (Волосы), поскольку если бы у жены к тому времени уже не было ее длинных волос, то действие не имело намеченного результата — сделать ей приятное, и аналогично для действия GiveChain.) На рис. 12.5, в показана декомпозиция этапа GiveComb с помощью метода Give... (OnCredit) (Получение... под залог). На первом этапе декомпозиции муж получает в свою собственность гребень и вручает его своей жене, дав обязательство передать часы в качестве оплаты в последующую дату. На втором этапе он вручает свои часы и выполняет данное обязательство. Декомпозиция этапа GiveChain осуществляется с помощью аналогичного метода. При условии, что оба этапа дарения упорядочиваются перед этапами доставки залога, такая декомпозиция становится решением задачи. (Обратите внимание на то, что теперь задача определена так, что супруги получат удовольствие от использования цепочки вместе с часами и гребнем вместе с волосами даже после того, как потеряют право владения на часы и волосы.)*

## Обсуждение вопроса

Вначале необходимо отметить определенные сложности: чистая задача планирования HTN неразрешима (если единственным допустимым способом уточнения

плана является декомпозиция), даже если соответствующее пространство состояний конечно! На первый взгляд такая ситуация может показаться весьма бесперспективной, поскольку основной смысл планирования HTN состоит в обеспечении высокой эффективности составления планов. Указанные сложности возникают потому, что декомпозиции действий могут оказаться **рекурсивными** (например, если выход на прогулку рассматривается как выполнение одного шага с последующим выходом на прогулку), поэтому планы HTN могут приобретать произвольную длину. В частности, даже кратчайшее решение HTN может оказаться неопределенно длинным, так что становится невозможным нахождение способа завершения поиска за какое-то постоянное время. Тем не менее существуют по меньшей мере три описанных ниже способа исправления указанного положения.

1. Исключить рекурсию, поскольку она действительно требуется лишь в очень немногих проблемных областях. В таком случае все планы HTN приобретают конечную длину и могут быть успешно исследованы.
2. Ограничить длину решений, которые нас интересуют. Поскольку пространство состояний является конечным, план, включающий больше этапов, чем имеется состояний в пространстве состояний, обязательно должен включать цикл, в котором неоднократно посещается одно и то же состояние. Мы ничего не потеряем, исключив решения HTN такого рода, поэтому следует контролировать длину поиска.
3. Принять гибридный подход, в котором сочетается планирование POP и HTN. Для определения того, существует ли план, достаточно применить планирование с частичным упорядочением, отдельно взятое, поэтому, безусловно, задача планирования с помощью гибридного подхода является разрешимой.

При использовании третьего метода необходимо соблюдать определенную осторожность. В планировании POP примитивные действия могут соединяться в цепочки произвольными способами, поэтому иногда приходится сталкиваться с такими решениями, которые очень трудно понять и которые не имеют такой аккуратной иерархической организации, как планы HTN. Приемлемым компромиссом является управление гибридным поиском таким образом, чтобы операции декомпозиции действий стали предпочтительными по сравнению с операциями добавления новых действий, хотя и не до такой степени, чтобы вырабатывались планы HTN произвольной длины, прежде чем появится возможность добавления каких-либо примитивных действий. Один из способов осуществления такого управления состоит в использовании функции стоимости, которая предоставляет благоприятные условия действиям, введенным путем декомпозиции; чем более благоприятными являются эти условия, тем больше поиск будет напоминать чистое планирование HTN и тем более иерархическим будет решение. Иерархические планы обычно намного проще для выполнения в реальных условиях, поэтому их легче исправить, если что-то при их осуществлении нарушается.

Еще одной важной характерной особенностью планов HTN является возможность совместного использования подзадач. Напомним, что *совместным использованием подзадач* называется применение одного и того же действия для реализации двух разных этапов в декомпозиции планов. Если совместное использование подзадач запрещено, то каждая конкретизация декомпозиции  $d'$  должна быть выполнена

только одним способом, а не многими, что приводит к отсечению значительной части пространства поиска. Обычно такое отсечение обеспечивает некоторую экономию времени и в худшем случае приводит к решению, лишь немного более длинному, чем оптимальное. Но в некоторых случаях возникают более существенные проблемы. Например, рассмотрим цель: “Насладись медовым месяцем и создай большую семью”. Библиотека планов может содержать решение “вступи в брак и отправляйся на Гавайи” для первой подцели и “вступи в брак и заведи двух детей” для второй. Без совместного использования подзадач план будет включать два разных действия по вступлению в брак для одного человека, но такой вариант многие считают весьма нежелательным.

Интересным примером анализа затрат и результатов совместного использования подзадач является применение этого метода при оптимизации компиляторов. Рассмотрим задачу компиляции выражения  $\tan(x) - \sin(x)$ . В большинстве компиляторов такая задача выполняется путем слияния результатов двух отдельных вызовов процедур тривиальным способом: все этапы процедуры вычисления тангенса  $\tan$  выполняются перед каким-либо из этапов вычисления синуса  $\sin$ . Но рассмотрим следующие аппроксимации для  $\sin$  и  $\tan$  с помощью рядов Тейлора:

$$\tan x \approx x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315}; \quad \sin x \approx x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040}$$

Планировщик HTN с совместным использованием подзадач может выработать более эффективное решение, поскольку он способен выбрать вариант реализации многочисленных этапов вычисления  $\sin$  в сочетании с существующими этапами вычисления  $\tan$ . В большинстве компиляторов межпроцедурное совместное использование этапов такого рода не осуществляется, поскольку для них потребовалось бы слишком много времени на рассмотрение всех возможных совместно используемых планов. Вместо этого в большинстве компиляторов каждый субплан вырабатывается независимо и только после этого, возможно, происходит модификация результата с помощью локального оптимизатора.

Если учесть все эти дополнительные сложности, вызванные введением декомпозиций действий, можно ли рассчитывать на то, что планирование HTN окажется эффективным? Фактические причины сложностей трудно проанализировать на практике, поэтому рассмотрим идеализированный случай. Предположим, например, что необходимо составить план с  $n$  действиями. Для неиерархического планировщика с прямым поиском в пространстве состояний при наличии  $b$  допустимых действий в каждом состоянии затраты составят  $O(b^n)$ . А применительно к планировщику HTN предположим, что структура декомпозиции является регулярной: каждое непримитивное действие имеет  $d$  возможных декомпозиций, каждая из которых сводится к  $k$  действиям на следующем, более низком уровне. Необходимо определить, сколько разных деревьев декомпозиции существует в этой структуре. Итак, если имеется  $n$  действий на примитивном уровне, то количество уровней ниже корня равно  $\log_k n$ , поэтому количество внутренних узлов декомпозиции определяется выражением  $1+k+k^2+\dots+k^{\log_k n-1} = (n-1)/(k-1)$ . Каждый внутренний узел имеет  $d$  возможных декомпозиций, поэтому существует  $d^{(n-1)/(k-1)}$  возможных регулярных деревьев декомпозиции, которые могут быть сформированы. Исследуя эту формулу, можно определить, что уменьшение значения  $d$  и увеличение значения  $k$  позволяет получить огромную экономию: по сути затраты измеряются корнем  $k$ -й степени от

стоимости неиерархического решения, если значения  $b$  и  $d$  являются сопоставимыми. С другой стороны, составление библиотеки планов, которая включает небольшое количество длинных декомпозиций, но тем не менее позволяет решить любую задачу, не всегда возможно. В другой формулировке эту мысль можно выразить так, что длинные декомпозиционные макроподстановки, применимые для решения широкого ряда задач, являются чрезвычайно ценными.

Еще одной и, возможно, более важной причиной, позволяющей убедиться в том, что планирование HTN эффективно, является то, что эти методы хорошо зарекомендовали себя на практике. Почти все планировщики для крупномасштабных приложений являются планировщиками HTN, поскольку планирование HTN позволяет людям-экспертам применять свои крайне важные знания о том, как следует выполнять сложные задания, чтобы можно было формировать большие планы с малыми вычислительными издержками. Например, система O-Plan [93], в которой планирование HTN сочетается с составлением расписаний, использовалась для разработки производственных планов в компании Hitachi. При этом типичная задача охватывала производственную линию с 350 различными изделиями, 35 сборочными агрегатами и с более чем 2000 различных операций. Планировщик O-Plan вырабатывает тридцатисуточные расписания с тремя восьмичасовыми сменами в сутки, включающие миллионы операций.

Таким образом, ключом к планированию HTN является составление библиотеки планов, в которой зашифрованы известные методы осуществления сложных, высокоуровневых действий. Одним из способов составления такой библиотеки является изучение требуемых методов на опыте решения задач. Накопив тяжелым трудом опыт планирования в ходе разработки какого-то плана с нуля, агент может сохранить этот план в библиотеке как метод осуществления высокоровневого действия, определяемого данным конкретным заданием. Таким образом, агент может становиться со временем все более и более компетентным по мере того, как будет находить новые методы на основе старых. Одной из важных характеристик такого процесса обучения является способность обобщать созданные методы, устранивая детали, характерные для данного экземпляра задачи (например, в задаче строительства дома таковыми являются имя подрядчика или адрес земельного участка), и сохраняя в плане только ключевую информацию. Методы осуществления такого рода обобщения описаны в главе 19. Авторы не могут себе даже представить, что человечество смогло бы достичь современного уровня компетентности без какого-то подобного механизма.

## 12.3. ПЛАНИРОВАНИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ В НЕДЕТЕРМИНИРОВАННЫХ ПРОБЛЕМНЫХ ОБЛАСТЯХ

До сих пор в этой главе рассматривались только проблемные области **классического планирования**, которые являются полностью наблюдаемыми, статическими и детерминированными. Кроме того, предполагалось, что описания действий являются правильными и полными. В таких обстоятельствах любой агент получает возможность вначале составлять план, а затем выполнять этот план буквально “с закрытыми глазами”. С другой стороны, в неопределенной среде агент должен использовать результаты своих восприятий для обнаружения того, что происходит в процессе вы-

полнения плана, а также, возможно, модифицировать или заменять этот план, если случается что-то непредвиденное.

Агентам приходится также иметь дело и с неполной, и с неправильной информацией. Неполнота возникает из-за того, что мир является частично наблюдаемым, или недетерминированным, или обладающим и тем и другим свойством. Например, дверца шкафа с канцелярскими принадлежностями может быть либо закрыта, либо не закрыта на замок; один из моих ключей может открыть или не открыть дверцу, если она на замке, а я могу знать или не знать, в чем имеющаяся у меня информация об этой ситуации является неполной. Поэтому моя модель мира является слабой, но правильной. С другой стороны, неправильность часто возникает, поскольку мир не всегда соответствует моей модели мира; например, я могу быть уверенным в том, что мой ключ откроет шкаф с канцелярскими принадлежностями, но окажусь неправ, если в нем сменили замок. Не обладая способностью справляться с неправильной информацией, агент становится таким же неинтеллектуальным, как и навозный жук (с. 81), который пытается заткнуть свою норку несуществующим шариком навоза даже после того, как шарик вынули из его лапок.

Возможность получения полной или правильной информации зависит от того, какой степенью недетерминированности характеризуется мир. При **ограниченной недетерминированности** действия могут иметь непредсказуемые результаты, но все возможные результаты можно перечислить в аксиомах описания действия. Например, при подбрасывании монеты вполне резонно считать, что результатом будет *Heads* (Орел) или *Tails* (Решка). Агент получает способность справляться с ограниченной недетерминированностью, составляя планы, применимые во всех возможных обстоятельствах. При **неограниченной недетерминированности**, с другой стороны, множество возможных предусловий или результатов либо неизвестно, либо слишком велико для того, чтобы в нем можно было выполнить полный поиск. Такие ситуации могут возникать в случае очень сложных или динамичных проблемных областей, таких как вождение автомобиля, экономическое планирование и разработка военной стратегии. Агенту удастся справиться с неограниченной недетерминированностью, только если он способен пересматривать свои планы и/или свою базу знаний. Неограниченная недетерминированность тесно связана с **проблемой спецификации** (qualification problem), описанной в главе 10, — с проблемой, определяемой невозможностью перечислить все предусловия, требуемые для того, чтобы какое-то действие в реальном мире имело свой намеченный результат.

Существуют четыре описанных ниже метода планирования для осуществления действий в условиях недетерминированности. Первые два из них применимы для ограниченной недетерминированности, а последние два — для неограниченной недетерминированности.

- **Планирование без использования датчиков.** Этот метод, называемый также **совместимым планированием**, предусматривает создание стандартных, последовательных планов, которые должны выполняться без учета результатов восприятия. Алгоритм планирования без использования датчиков должен обеспечивать, чтобы цель достигалась в плане при всех возможных обстоятельствах, независимо от истинного начального состояния и фактических результатов действий. Планирование без использования датчиков основано на идее **принуждения**, согласно которой мир может быть принудительно переве-

ден в данное конкретное состояние, даже если агент обладает лишь частичной информацией о текущем состоянии. Принуждение не всегда возможно, поэтому планирование без использования датчиков часто является неприменимым. Методы решения задач без использования датчиков, предусматривающие поиск в пространстве доверительных состояний, были описаны в главе 3.

- **Условное планирование.** В этом подходе, называемом также **планированием с учетом непредвиденных ситуаций**, действия в условиях ограниченной недетерминированности осуществляются путем создания условного плана с различными ответвлениями для самых разных непредвиденных ситуаций, какие только могут возникнуть. Так же как и в классическом планировании, агент вначале составляет план, а затем выполняет подготовленный план. Агент определяет, какая часть плана должна быть выполнена, включив в план **действия по восприятию** для проверки соответствующих условий. Например, в проблемной области воздушных перевозок могут быть предусмотрены планы, в которых указано “проверить, работает ли аэропорт *SFO* (Сан-Франциско); в случае положительного ответа полететь в этот аэропорт, в противном случае полететь в *Окленд*”. Условное планирование рассматривается в разделе 12.4.
- **Контроль выполнения и перепланирование.** При этом подходе агент может использовать любой из описанных выше методов планирования (классический, без использования датчиков или условный) для формирования плана, но использует также **контроль выполнения** для оценки того, предусмотрена ли в плане фактически сложившаяся текущая ситуация, или план должен быть пересмотрен. **Перепланирование** осуществляется, если что-то происходит не так, как надо. Благодаря этому агент приобретает способность справляться с неограниченной недетерминированностью. Например, даже если агент, способный к перепланированию, не предвидел возможность закрытия аэропорта *SFO*, то может обнаружить эту ситуацию после ее возникновения и снова вызвать планировщик для поиска нового пути к цели. Перепланирующие агенты рассматриваются в разделе 12.5.
- **Непрерывное планирование.** Все планировщики, рассматривавшиеся до сих пор, спроектированы так, что они достигают цели, а затем останавливаются, а непрерывный планировщик предназначен для того, чтобы заниматься планированием в течение всего срока своего существования. Он способен справляться с непредвиденными ситуациями в своей среде, даже если они возникают в ходе того, как агент занимается составлением плана. Он способен также обеспечить отказ от целей и создание дополнительных целей с помощью **формулировки цели**. Непрерывное планирование описано в разделе 12.6.

Рассмотрим один пример, позволяющий выяснить различия между агентами разных типов. Задача состоит в следующем: дано начальное состояние, в котором имеются стул, стол и несколько банок с краской, притом что все они имеют неизвестный цвет; достичь состояния, в котором стул и стол имеют одинаковый цвет.

**Классический планирующий** агент не может справиться с этой задачей, поскольку начальное состояние задано не полностью, — мы не знаем, какой цвет имеет мебель.

Агент, осуществляющий **планирование без использования датчиков**, должен найти план, который работает, не требуя применения каких-либо датчиков во время его

выполнения. Решение состоит в том, чтобы открыть любую банку с краской и нанести эту краску и на стул, и на стол, тем самым **принудительно переведя** их в такое состояние, в котором они имеют одинаковый цвет (даже несмотря на то, что агент так и не узнает, каким стал этот цвет). Принуждение является наиболее приемлемым, если обработка высказываний требует больших затрат или не существует возможности обеспечить восприятие. Например, врачи часто предписывают антибиотик с широким спектром действия, а не используют условный план, согласно которому нужно выполнить анализ крови, затем дождаться получения результатов и только после этого выписать более специализированный антибиотик. Они принимают такое решение потому, что задержки и затраты, связанные с выполнением анализа крови, обычно слишком велики.

Агент, занимающийся **условным планированием**, может создать лучший план. Он вначале определяет цвет стола и стула; если этот цвет является одинаковым, то план выполнен. В противном случае агент просматривает надписи на банках с краской; если имеется банка с краской такого же цвета, как и у одного из предметов мебели, то агент наносит эту краску и на другой предмет. Если же оба эти предположения оказываются недействительными, агент окрашивает и стол, и стул произвольно выбранной краской.

**Перепланирующий** агент может выработать тот же план, что и условный планировщик, или вначале предусмотреть меньше ответвлений и по мере необходимости заполнять остальные фрагменты плана в ходе его выполнения. Он также способен справляться с неточностями в описании его действий. Например, предположим, что действие *Paint(obj, color)* рассматривается как имеющее детерминированный результат *Color(obj, color)*. Условный планировщик будет просто считать, что желаемый результат достигнут непосредственно после выполнения действия, а перепланирующий агент способен проверить результат, и если он не был истинным (возможно, потому, что агент допустил небрежность и не прокрасил полностью какой-то участок), то имеет возможность затем перепланировать свои действия, чтобы снова покрасить дефектный участок. Мы вернемся к данному примеру на с. 596.

**Непрерывно планирующий** агент, кроме учета непредвиденных событий, способен пересматривать свои планы должным образом, если, допустим, мы внесем дополнительную цель, чтобы сегодняшний обед прошел еще за этим столом, поэтому нужно будет отложить выполнение плана окрашивания.

В реальном мире в агентах применяется сочетание нескольких подходов. Производители автомобилей продают запасные колеса и надувные подушки безопасности, которые представляют собой физические воплощения ответвлений условного плана, позволяющих справляться с такими ситуациями, как проколы колес или аварии; с другой стороны, большинство водителей автомобилей никогда не учитывают такие возможности, поэтому отвечают на проколы и аварии как перепланирующие агенты. Как правило, агенты создают условные планы только для таких непредвиденных ситуаций, которые имеют важные последствия, а шансами на то, что дела пойдут не лучшим образом, нельзя пренебречь. Таким образом, водителю автомобиля, принимающему поездку через пустыню Сахара, придется явно учесть возможность поломок в пути, тогда как поездка в соседний супермаркет требует гораздо меньшего злаговременного планирования.

Агенты, описанные в этой главе, предназначены для осуществления действий в условиях недетерминированности, но не обладают способностью учитывать ком-

промиссы между вероятностью успеха и стоимостью составления плана. В главе 16 рассматриваются дополнительные инструментальные средства, позволяющие справляться с этими проблемами.

## 12.4. УСЛОВНОЕ ПЛАНИРОВАНИЕ

Условное планирование представляет собой один из способов учета неопределенности путем проверки того, что фактически происходит в среде при выполнении заранее заданных пунктов плана. Условное планирование можно проще всего объяснить на примере полностью наблюдаемых вариантов среды, поэтому начнем его описание с такого случая. Случай с частично наблюдаемой средой является более сложным, но и более интересным.

### Условное планирование в полностью наблюдаемых вариантах среды

Полная наблюдаемость означает, что агент всегда знает текущее состояние. Но если среда является недетерминированной, то агент не будет способен предвидеть результат своих действий. Агент, занимающийся условным планированием, преодолевает такую недетерминированность, встраивая в свой план (на этапе планирования) условные этапы, в которых проверяется состояние среды (на этапе выполнения), для определения того, что делать дальше. Поэтому проблема состоит в том, как создавать такие условные планы.

Мы будем использовать в качестве примера проблемной области почтенный **мир пылесоса**, пространство состояний которого для данного детерминированного случая описано на с. 117. Напомним, что доступными действиями являются *Left*, *Right* и *Suck*. Нам потребуются определенные высказывания для описания этих состояний. Допустим, что предикат *AtL* (*AtR*) будет истинным, если агент находится в левом (правом) квадрате<sup>5</sup>, а предикат *CleanL* (*CleanR*) истинен, если левый (правый) квадрат чист. Прежде всего необходимо дополнить язык *Strips* для того, чтобы он позволял учитывать недетерминированность. Для этого предусмотрено, чтобы действия имели **дизъюнктивные результаты**, а это означает, что действие может иметь два или несколько различных результатов при каждом его выполнении. Например, допустим, что действие по перемещению *Left* (Влево) иногда оканчивается неудачей. В таком случае следующее обычное описание действия:

*Action(Left, Precond: AtR, Effect: AtL  $\wedge$   $\neg$ AtR)*

должно быть модифицировано так, чтобы в него был включен следующий дизъюнктивный результат:

*Action (Left, Precond: AtR, Effect: AtL  $\vee$   $\neg$ AtR) (12.1)*

Авторы сочли также целесообразным, чтобы действия имели **условные результаты**, когда результат действия зависит от состояния, в котором оно выполняется. Условные результаты присутствуют в слоте *Effect* действия и имеют синтаксис

<sup>5</sup> Очевидно, что предикат *AtR* является истинным тогда и только тогда, когда предикат  $\neg$ AtL является истинным, и наоборот. Мы используем два высказывания в основном для того, чтобы улучшить удобство чтения.

“**when** *<condition>*: *<effect>*”. Например, чтобы промоделировать действие *Suck*, необходимо записать следующее:

```
Action(Suck, Precond:, Effect: (when AtL: CleanL) ∧
(when AtR: CleanR))
```

Условные результаты не вносят недетерминированность, но позволяют ее моделировать. Например, предположим, что в нашем распоряжении имеется не совсем исправный пылесос, который иногда оставляет мусор в квадрате назначения в процессе своего передвижения, но только если этот квадрат чист. Такую ситуацию можно промоделировать с помощью следующего описания, которое одновременно является и дизъюнктивным, и условным<sup>6</sup>:

```
Action(Left, Precond: AtR, Effect: AtL ∨ (AtL ∧ when CleanL: ¬CleanL))
```

Для создания условных планов требуются **условные этапы**. Мы будем записывать их с использованием синтаксиса “**if** *<test>* **then** *plan<sub>A</sub>* **else** *plan<sub>B</sub>*”, где *<test>* — булева функция переменных состояния. Например, условным этапом для мира пылесоса может быть следующий: “**if** AtL ∧ CleanL **then** Right **else** Suck”. Выполнение такого этапа осуществляется очевидным способом. В результате вложения условных этапов линейные планы превращаются в древовидные.

Нам требуются условные планы, которые работают невзирая на то, какие результаты действий фактически будут получены. Такая проблема уже встречалась раньше в данной книге, но немного в другом виде. В играх с двумя игроками (см. главу 6) была поставлена задача найти ходы, которые приводят к выигрышу независимо от того, какие ходы делает противник. По этой причине задачи недетерминированного планирования часто называют **играми против природы**.

Рассмотрим конкретный пример из мира пылесоса. В начальном состоянии робот-агент находится в правом квадрате мира, в котором нет мусора; поскольку среда полностью наблюдаема, агент знает полное описание состояния, AtR ∧ CleanL ∧ CleanR. В целевом состоянии робот должен находиться в левом квадрате мира, в котором нет мусора. Такая задача была бы совершенно тривиальной, если бы пылесос не подчинялся “двойному закону Мэрфи” и не оставлял иногда мусор при его перемещении в чистый квадрат назначения, а иногда — при выполнении действия *Suck* в чистом квадрате.

“Дерево игры” для этой среды показано на рис. 12.6. Действия выполняются роботом в узлах “состояния” этого дерева, а природа решает, каким должен быть результат в узлах “жеребьевки”, обозначенных кружками. Решением является поддерево, в котором, во-первых, рядом с каждым листовым узлом имеется целевой узел, во-вторых, задается одно действие в каждом из узлов “состояния” и, в третьих, включена каждая ветвь результата в каждом из узлов “жеребьевки”. Решение на этом рисунке показано жирными линиями; оно соответствует плану [*Left*, **if** AtL ∧ CleanL ∧ CleanR **then** [] **else** *Suck*]. (На данное время, поскольку используемый

---

<sup>6</sup> Условный результат **when** CleanL: ¬ CleanL может выглядеть немного странным. Но следует учитывать, что здесь CleanL относится к ситуации до выполнения действия, а ¬CleanL — к ситуации после выполнения действия.

зуется планировщик в пространстве состояний, проверки в условных этапах приводят к получению полных описаний состояния.)

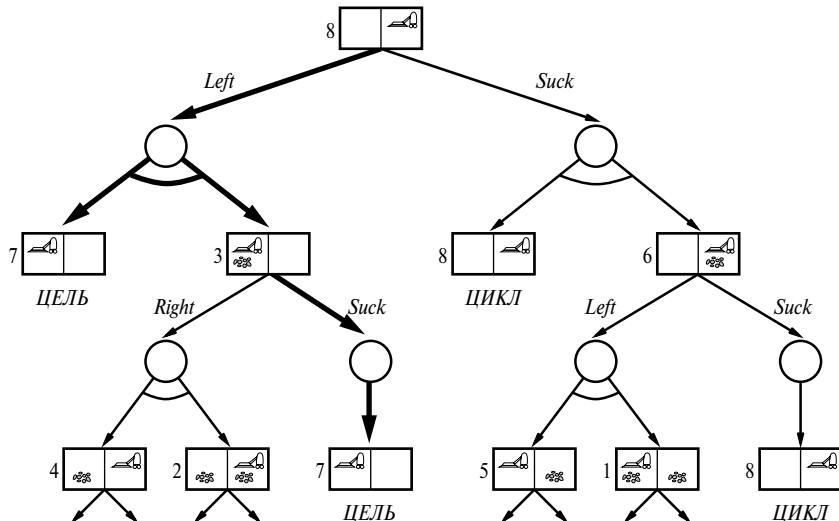


Рис. 12.6. Первые два уровня дерева поиска для мира пылесоса с "двойным законом Мэрфи". Узлами состояния являются узлы OR, в которых должно быть выбрано некоторое действие. Узлы жеребьевки, показанные в виде кружков, являются узлами AND, для которых требуется учитывать все результаты, как показывает кривая линия, соединяющая исходящие ветви. Решение показано жирными линиями

Для получения точных решений в играх используется **алгоритм минимакса** (см. листинг 6.1), а для условного планирования обычно применяются две его модификации. Во-первых, узлы MAX и MIN могут быть преобразованы в узлы OR и AND. Интуитивно кажется очевидным, что в плане требуется предпринимать некоторые действия в каждом достигнутом в нем состоянии, но должны учитываться все результаты предпринятого действия. Во-вторых, алгоритм должен возвращать условный план, а не просто отдельный ход. В узле OR план состоит в выборе действия, за которым следует то, что должно произойти. А в узле AND план представляет собой вложенный ряд этапов "if-then-else" с определением субпланов для каждого результата; проверки в этих этапах возвращают полные описания состояния<sup>7</sup>.

Формально говоря, определенное нами пространство поиска представляет собой граф AND-OR. В главе 7 графы AND-OR рассматривались в контексте пропозиционального вывода на основе хорновских выражений, а в этой главе ветвями являются действия, а не этапы логического вывода, но алгоритм остается тем же самым. В листинге 12.4 приведен рекурсивный алгоритм поиска в глубину, предназначенный для поиска в графе AND-OR.

Одной из ключевых особенностей этого алгоритма является тот способ, с помощью которого он справляется с циклами, часто возникающими в задачах недетерминированного планирования (например, если какое-то действие иногда не имеет

<sup>7</sup> Такие планы можно также составлять с использованием конструкции case.

результата или если может быть исправлен нежелательный результат). Если текущее состояние идентично одному из состояний в пути от корня, то алгоритм выполняет возврат с индикатором неудачи. Это не означает, что нет решения, достижимого из текущего состояния; просто такая ситуация свидетельствует о том, что если есть нециклическое решение, оно должно быть достижимым из более раннего воплощения текущего состояния, поэтому его новое воплощение может быть отброшено. С помощью этой проверки можно гарантировать, что алгоритм завершит свою работу в любом конечном пространстве состояний, поскольку каждый путь должен достигнуть цели, тупика или повторяющегося состояния. Обратите внимание на то, что в алгоритме не осуществляется проверка, не является ли текущее состояние повторением какого-то состояния в каком-то другом пути от корня; этот вопрос исследуется в упр. 12.15.

**Листинг 12.4. Алгоритм поиска в графах AND-OR, сформированных в условиях недетерминированных вариантов среды.** Предполагается, что функция `Successors` возвращает список действий, каждое из которых связано с множеством возможных результатов. Задача состоит в том, чтобы найти условный план, который достигает целевого состояния во всех обстоятельствах

---

```

function And-Or-Graph-Search(problem) returns условный план
    или индикатор неудачи failure
    Or-Search(Initial-State[problem], problem, [])

function Or-Search(state, problem, path) returns условный план
    или индикатор неудачи failure
    if Goal-Test[problem](state) then return пустой план
    if состояние state входит в состав path then return failure
    for each action, state_set in Successors[problem] (state) do
        plan  $\leftarrow$  And-Search(state_set, problem, [state | path])
        if plan  $\neq$  failure then return [action | plan]
    return failure

function And-Search(state_set, problem, path) returns условный план
    или индикатор неудачи failure
    for each si in state_set do
        plani  $\leftarrow$  Or-Search(si, problem, path)
        if plani = failure then return failure
    return [if s1 then plan1 else if s2 then plan2 else ...
            if sn-1 then plann-1 else plann]

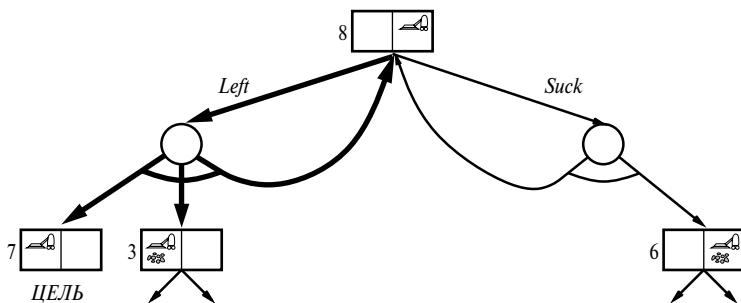
```

---

Планы, возвращаемые алгоритмом And-Or-Graph-Search, содержат условные шаги, в которых проверяется описание всего состояния для выбора следующей ветви. Но во многих случаях можно справиться с этой работой с помощью менее исчерпывающих проверок. Например, план решения на рис. 12.6 может быть записан просто как [*Left*, **if** *CleanL* **then** [] **else** *Suck*]. Это связано с тем, что достаточно провести единственную проверку, *CleanL*, для разделения состояния в узле AND на два одноэлементных множества, чтобы после проверки агент мог точно определить, в каком состоянии он находится. В действительности проведение ряда проверок по принципу “if-then-else” отдельных переменных всегда позволяет разделить множество состояний на одноэлементные множества, при условии, что среда является полностью наблюдаемой. Поэтому можно без потери общности ограничиться проведением проверок отдельных переменных.

В недетерминированных проблемных областях часто возникает еще одна, последняя сложность: не всегда удается выполнить текущую операцию с первого раза, поэтому приходится предпринимать еще одну попытку. Например, рассмотрим пылесос в мире с “тройным законом Мэрфи”, который (кроме ранее указанных его недостатков) иногда отказывается переходить туда, куда требует переданная ему команда, например, действие *Left* может иметь дизъюнктивный результат  $AtL \vee AtR$ , как в уравнении 12.1. Теперь уже нельзя гарантировать, что план `[Left, if CleanL then [] else Suck]` будет работать. Часть нового графа поиска показана на рис. 12.7 ; очевидно, что больше нет каких-либо нециклических решений и алгоритм And-Or-Graph-Search выполняет возврат с индикатором неудачи. Тем не менее существует **циклическое решение**, которое заключается в том, что нужно продолжать попытки выполнять действие *Left* до тех пор, пока одна из попыток не достигнет успеха. Это решение можно выразить, введя **метку** для обозначения некоторой части плана и используя эту метку позднее, вместо повторения самого плана. Таким образом, соответствующее циклическое решение может выглядеть таким образом:

[ $L_1$ : Left, **if**  $AtR$  **then**  $L_1$  **else if**  $CleanL$  **then** [] **else** Suck]



*Рис. 12.7. Первый уровень графа поиска для мира пылесоса, подчиняющегося “тройному закону Мэрфи”, где циклы показаны явно. Все решения для данной задачи представляют собой циклические планы*

(Более удобный синтаксис для циклической части этого плана мог бы предусматривать применение конструкции “**while** *AtR do Left*”.) Изменения, необходимые для внесения в алгоритм And-Or-Graph-Search, рассматриваются в упр. 12.16. Из этого следует основной вывод, что цикл в пространстве состояний, ведущий в состояние  $L$ , преобразуется в цикл в плане, ведущий в ту точку, где выполнялся субплан для состояния  $L$ .

Теперь мы получили возможность синтезировать сложные планы, которые во многом выглядят как программы, с условными выражениями и циклами. К сожалению, эти циклы, в принципе, могут стать бесконечными. Например, в представлении действия для мира с “тройным законом Мэрфи” ничего не сказано, что действие *Left* в конечном итоге закончится успешно. Поэтому циклические планы не столь желательны, как ациклические, но вполне могут рассматриваться как решения, при условии, что каждый листовой узел представляет собой целевое состояние и любой листовой узел достижим из каждой точки в плане.

## Условное планирование в частично наблюдаемых вариантах среды

В предыдущем разделе рассматривались полностью наблюдаемые варианты среды, преимущество которых состоит в том, что во время условных проверок можно задавать любые вопросы и быть уверенным в том, что будет получен ответ. Но в реальном мире гораздо чаще встречается частичная наблюдаемость. В начальном состоянии частично наблюдаемой задачи планирования агент обладает лишь некоторым объемом знаний о действительном состоянии. Простейший способ промоделировать такую ситуацию состоит в том, чтобы принять предположение, что начальное состояние принадлежит к **множеству состояний**; множество состояний представляет собой способ описания начального **доверительного состояния** агента<sup>8</sup>.

Предположим следующее: агенту в мире пылесоса известно, что он находится в правом квадрате и что этот квадрат чист, но он не может определить с помощью датчиков наличие или отсутствие мусора в других квадратах. В таком случае, насколько известно агенту, он может находиться в одном из двух состояний: левый квадрат может быть либо чистым, либо грязным. Это доверительное состояние обозначено на рис. 12.8 буквой А. На этом рисунке показана часть графа AND-OR для мира пылесоса с “альтернативным двойным законом Мэрфи”, в котором мусор может иногда оставаться сзади, после того как агент покидает чистый квадрат<sup>9</sup>. Если бы этот мир был полностью наблюдаемым, то агент имел бы возможность сформировать циклическое решение в такой форме: “Продолжать двигаться влево и вправо, всасывая мусор везде, где он появляется, до тех пор, пока оба квадрата не станут чистыми, а я не буду находиться в левом квадрате” (см. упр. 12.16). К сожалению, при использовании лишь локального датчика грязи этот план является невыполнимым, поскольку невозможно определить истинностное значение проверки “оба квадрата стали чистыми”.

Рассмотрим, как формируется граф AND-OR. Из доверительного состояния А мы показываем результат перемещения с помощью действия *Left* (другие действия не имеют смысла). Поскольку агент может оставить за собой мусор, эти два возможных начальных состояния мира становятся четырьмя возможными состояниями, как показано в прямоугольниках В и С. Эти состояния формируют два различных доверительных состояния, которые классифицируются по доступной информации датчика<sup>10</sup>. В доверительном состоянии В агент имеет информацию *CleanL*, а в доверительном состоянии С — информацию *¬CleanL*. В результате уборки мусора в состоянии С агент переходит в состояние В. После перехода с помощью действия *Right* из состояния В агент может оставить или не оставить за собой мусор, поэтому снова возникают четыре возможные состояния мира, которые подразделяются в соответствии со знаниями агента о том, является ли правый квадрат чистым, *CleanR* (возврат в состояние А), или грязным, *¬CleanR* (переход в доверительное состояние Д).

<sup>8</sup> Эти понятия представлены в разделе 3.6, к которому читателю может потребоваться обратиться еще раз, прежде чем изучать дальнейший материал.

<sup>9</sup> Этот феномен должен быть знаком родителям с маленькими детьми. Как обычно, приносим свои извинения тем, кто не является таковыми.

<sup>10</sup> Обратите внимание на то, что эти состояния не классифицируются по тому признаку, остается ли сзади грязь после передвижения агента. Ветвление в пространстве доверительного состояния вызвано альтернативными результатами изменения знаний, а не альтернативными результатами физических изменений.

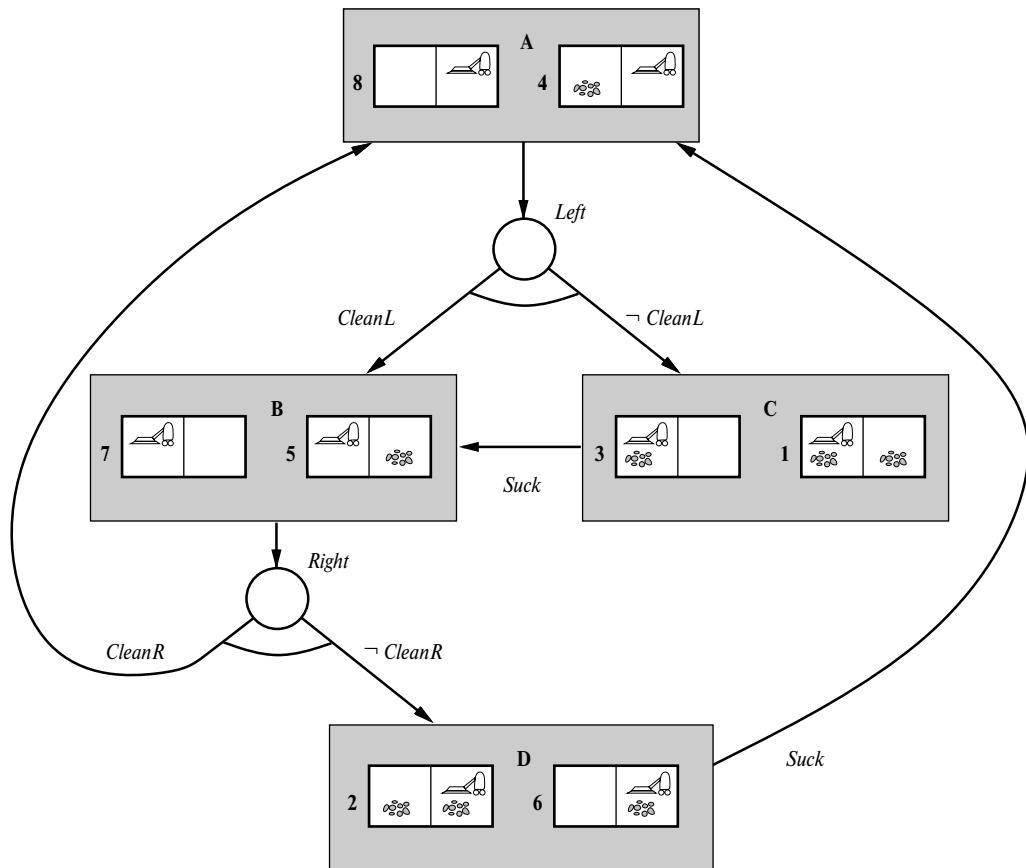


Рис. 12.8. Часть графа AND-OR для мира пылесоса с “альтернативным двойным законом Мэрфи”, в котором грязь иногда может оставаться сзади, после того как агент покидает чистый квадрат. Агент не может получать с помощью датчиков информацию о наличии грязи в других квадратах

Подводя итог, можно отметить, что недетерминированные частично наблюдаемые варианты среды приводят к получению графа AND-OR доверительных состояний. Поэтому для поиска условных планов может применяться точно такой же алгоритм, как и в случае полностью наблюдаемой среды, а именно And-Or-Graph-Search. Еще один способ понять, что происходит, состоит в следующем: необходимо отметить, что доверительное состояние агента всегда является полностью наблюдаемым, т.е. агент всегда знает о том, что он знает. Решение “стандартной” задачи в полностью наблюдаемой среде представляет собой частный случай, в котором каждое доверительное состояние является одноэлементным множеством, содержащим одно и только одно физическое состояние.

Можем ли мы на этом завершить описание данной темы? Не совсем! Еще необходимо определить, как должны быть представлены доверительные состояния, как работают средства сбора информации (датчики) и как в этой новой постановке задачи должны записываться описания действий.

Для описания доверительных состояний могут применяться три описанных ниже основных варианта.

1. Множества полных описаний состояния. Например, начальное доверительное состояние на рис. 12.8 может быть описано следующим образом:

$$\{ (AtR \wedge CleanR \wedge CleanL), (AtR \wedge CleanR \wedge \neg CleanL) \}$$

Это представление является простым в использовании, но очень дорогостоящим: если имеется  $n$  булевых высказываний, определяющих состояние, то доверительное состояние может содержать  $O(2^n)$  описаний физических состояний, каждый из которых имеет размер  $O(n)$ . А если агент знает только небольшую часть этих высказываний, то могут возникать экспоненциально большие доверительные состояния, поскольку, чем меньше знает агент, тем больше количества возможных состояний, в которых он может находиться.

2. Логические высказывания, которые точно представляют множество возможных миров в доверительном состоянии. Например, определение начального состояния можно записать следующим образом:

$$AtR \wedge CleanR$$

Очевидно, что каждое доверительное состояние можно представить с помощью одного и только одного логического высказывания; в случае необходимости можно было бы использовать дизъюнкцию всех конъюнктивных описаний состояния, но рассматриваемый пример показывает, что могут существовать более компактные высказывания.

Один из недостатков общих логических высказываний состоит в том, что может существовать много различных, логически эквивалентных высказываний, которые описывают одно и то же доверительное состояние, поэтому проверка повторяющихся состояний в алгоритме поиска в графе может потребовать применения средств общего доказательства теорем. По этой причине желательно было бы использовать для высказываний каноническое представление, в котором каждое доверительное состояние соответствует одному и только одному высказыванию<sup>11</sup>. В одном из таких представлений используется конъюнкция литералов, упорядоченных по именам термов; в качестве примера можно привести  $AtR \wedge CleanR$ . Такое представление можно рассматривать как стандартное представление состояния с учетом **предположения об открытом мире**, описанного в главе 11. Не все логические высказывания могут быть записаны в указанной форме (например, не существует способа представления высказывания  $AtL \vee CleanR$ ), но такой метод представления доверительных состояний может применяться во многих проблемных областях.

3. Высказывания с оценкой знаний, описывающие знания агента (см. также раздел 7.7). Для данного начального состояния такое высказывание имеет следующий вид:

$$K(AtR) \wedge K(CleanR)$$

---

<sup>11</sup> Наиболее известным каноническим представлением для пропозиционального высказывания общего вида являются **бинарные диаграммы решения**, или BDD (binary decision diagram) [200].

где  $K$  обозначает “знает, что”, а  $K(P)$  указывает, что агент знает<sup>12</sup> об истинности высказывания  $P$ . В сочетании с высказываниями с оценкой знаний используется **предположение о замкнутом мире** — если высказывания с оценкой знаний нет в списке, оно рассматривается как ложное. Например, в приведенном выше высказывании неявно присутствуют термы  $\neg K(CleanL)$  и  $\neg K(\neg CleanL)$ , поэтому оно представляет тот факт, что агент не знает истинностного значения терма  $CleanL$ .

Как оказалось, второй и третий из описанных выше вариантов являются приблизительно эквивалентными, но мы будем использовать третий вариант — высказывания с оценкой знаний, поскольку они позволяют получить более реалистичное описание процесса получения информации с помощью датчиков и поскольку нам уже известно, как составлять описания Strips при использовании предположения о замкнутом мире.

В обоих вариантах каждый пропозициональный символ может быть представлен в одном из трех видов: положительный, отрицательный или неизвестный. Поэтому существует точно  $3^n$  возможных доверительных состояний, которые могут быть описаны таким образом. С другой стороны, множество доверительных состояний представляет собой показательное множество (множество всех подмножеств) множества физических состояний. Количество физических состояний равно  $2^n$ , поэтому количество доверительных состояний равно  $2^{2^n}$ , что намного превышает  $3^n$ ; это означает, что варианты 2 и 3 являются чрезвычайно ограниченными с точки зрения представления доверительных состояний. В настоящее время такая ситуация считается неизбежной, поскольку ~~любая схема, способная представить любое возможное доверительное состояние, в наихудшем случае для представления каждого из них потребует  $O(\log_2 2^{2^n}) = O(2^n)$  битов~~. Наши простые схемы требуют только  $O(n)$  битов для представления каждого доверительного состояния, поэтому позволяют достичь компромисса, в котором выразительность принесена в жертву компактности. В частности, если происходит некоторое действие, одно из предусловий которого неизвестно, то результирующее доверительное состояние не будет точно представимым и результат действия также станет неизвестным.

Теперь необходимо решить, как должны действовать средства сбора информации с помощью датчиков. В данном случае имеются два варианта. Можно применить ~~автоматический сбор информации с помощью датчиков~~, который означает, что на каждом временном этапе агент получает все доступные данные восприятия. В примере, приведенном на рис. 12.8, предполагается, что с помощью датчиков происходит автоматический сбор информации о местонахождении агента и наличии мусора в локальном квадрате. Еще один вариант состоит в том, что можно настаивать на использовании ~~средств активного сбора информации с помощью датчиков~~, а это означает, что данные восприятия приобретаются только в результате выполнения конкретных ~~действий по сбору информации с помощью датчиков~~, таких как

---

<sup>12</sup> Это — такая же система обозначений, которая использовалась для агентов на основе логических схем, описанных в главе 7. Некоторые авторы применяют ее для передачи такого смысла, как “знает о том, является ли истинным высказывание  $P$ ”. Задача определения преобразования между этими двумя представлениями является несложной.

*CheckDirt* и *CheckLocation*. Мы будем по очереди рассматривать каждый метод сбора информации с помощью датчиков.

Теперь сформулируем одно из описаний действия с использованием высказываний о знаниях. Предположим, что агент в мире “альтернативного двойного закона Мэрфи” перемещается с помощью действия *Left*, применяя автоматические средства сбора информации о наличии мусора в локальном квадрате; согласно правилам для этого мира, агент может оставить или не оставить за собой мусор, если квадрат был чистым. Этот результат, рассматриваемый как изменение физического состояния, должен быть дизъюнктивным, но если он рассматривается как результат изменения знаний, то просто сводится к тому, что у агента удаляются знания об истинности терма *CleanR*. Агент должен также знать, является ли истинным терм *CleanL*, получив эти знания тем или иным способом, поскольку для сбора информации о наличии мусора в локальном квадрате предусмотрены датчики; кроме того, агент знает, что он находится в квадрате *AtL*. Вся эта информация записывается следующим образом:

```
Action (Left, Precond: AtR
      Effect: K(AtL) ∧ ¬K(AtR) ∧ when CleanR: ¬K(CleanR) ∧
              when CleanL: K(CleanL) ∧
              when ¬CleanL: K(¬CleanL))
```

(12.2)

Обратите внимание на то, что предусловия и условия **when** представляют собой простые высказывания, а не высказывания с оценкой знаний. Так и должно быть, поскольку результаты действий зависят от фактического состояния мира, но как мы могли бы проверить истинность этих условий, если все, что мы знаем, представляет собой доверительное состояние? Если агент знает истинность высказывания, например,  $K(AtR)$  в текущем доверительном состоянии, то это высказывание должно быть истинным в текущем физическом состоянии и, безусловно, соответствующее действие является применимым. Если же агент не знает истинность какого-то высказывания (например, условия *CleanL* конструкции **when**), то данное доверительное состояние должно включать миры, в которых терм *CleanL* является истинным, и миры, в которых *CleanL* является ложным. Именно поэтому в результате данного действия возникает множество доверительных состояний. Таким образом, если начальным состоянием является  $(K(AtR) \wedge K(CleanR))$ , то после движения *Left* двумя результирующими доверительными состояниями становятся  $(K(AtL) \wedge K(CleanL))$  и  $(K(AtL) \wedge K(\neg CleanL))$ . В обоих случаях истинностное значение терма *CleanL* известно, поэтому в плане может использоваться проверка терма *CleanL*.

При использовании активных средств сбора информации с помощью датчиков (в отличие от автоматических средств сбора информации с помощью датчиков) агент получает новые данные восприятия только после выдачи запроса на их получение. Таким образом, после перемещения с помощью действия *Left* агент не знает, является ли левый квадрат грязным, поэтому два последних условных результата больше не появляются в описании действия в уравнении 12.2. Чтобы узнать, есть ли в квадрате мусор, агент может выполнить действие *CheckDirt* следующим образом:

```
Action (CheckDirt, Effect: when AtL ∧ CleanL: K(CleanL) ∧
        when AtL ∧ ¬CleanL: K(¬CleanL) ∧
        when AtR ∧ CleanR: K(CleanR) ∧
        when AtR ∧ ¬CleanR: K(¬CleanR))
```

(12.3)

Можно легко показать, что выполнение действия *Left*, за которым следует действие *CheckDirt*, при организации работы с помощью активных средств сбора информации приводит к получению тех же двух доверительных состояний, к которым приводило действие *Left* при использовании организации работы с помощью средств автоматического сбора информации. При активном сборе информации всегда имеет место то, что физические действия отображают доверительное состояние в единственное доверительное состояние-преемник. Многочисленные доверительные состояния могут быть введены только с помощью действий по сбору информации датчиками, которые позволяют получить конкретные знания и поэтому дают возможность использовать в планах условные проверки.

Выше был описан общий подход к условному планированию на основе поиска в пространстве состояний AND-OR. Такой подход зарекомендовал себя как чрезвычайно эффективный применительно к некоторым контрольным задачам, но другие задачи оказались трудноразрешимыми. Теоретически можно доказать, что условное планирование принадлежит к более трудному классу сложности, чем классическое планирование. Напомним, что определение класса задач NP состоит в том, что потенциальное решение может быть проверено для определения того, действительно ли оно является решением, за полиномиальное время. Это определение относится к классическим планам (по крайней мере, к планам, имеющим полиномиальные размеры), поэтому задача классического планирования относится к числу NP-трудных. Но в условном планировании потенциальное решение должно быть проверено для определения того, что для всех возможных состояний существует некоторый путь через план, позволяющий достичь цели. Проверка такой комбинации “все/некоторые” не может быть выполнена за полиномиальное время, поэтому условное планирование труднее, чем NP. Единственный способ выхода из этой ситуации состоит в том, чтобы игнорировать некоторые из возможных непредвиденных ситуаций, которые могут рассматриваться на этапе планирования, и реагировать на них, только когда они действительно возникают. Именно этот подход рассматривается в следующем разделе.

## **12.5. КОНТРОЛЬ ВЫПОЛНЕНИЯ И ПЕРЕПЛАНИРОВАНИЕ**

---

Агент, **контролирующий выполнение**, проверяет свои восприятия для определения того, все ли идет в соответствии с планом. Закон Мэрфи говорит нам о том, что даже самые тщательно продуманные планы, которые составляют мыши, люди и агенты, занимающиеся условным планированием, часто оканчиваются неудачей. Проблема заключается в неограниченной недетерминированности — всегда могут возникнуть непредвиденные обстоятельства, для которых описания действий, подготовленные агентом, будут неправильными. Поэтому в реальных вариантах среды всегда требуется контроль выполнения. Мы будем рассматривать два способа организации контроля выполнения: простую, но слабую форму, называемую **контролем действий**, в которой агент проверяет среду для определения того, что следующее действие окажется применимым, и более сложную, но и более эффективную форму, называемую **контролем плана**, в которой агент проверяет весь оставшийся план.

**Перепланирующий** агент знает, что делать, когда происходит что-то непредвиденное: снова вызвать планировщик, чтобы он предложил ему новый план достижения

цели. Для предотвращения использования слишком больших затрат времени на планирование такая операция обычно осуществляется в виде попытки исправить старый план — найти способ перехода из текущего непредвиденного состояния обратно в одно из тех состояний, которые были предусмотрены в плане.

В качестве примера еще раз рассмотрим мир пылесоса с “двойным законом Мэрфи”, показанный на рис. 12.6. В этом мире перемещение в чистый квадрат иногда приводит к тому, что в этом квадрате откладывается мусор, но что если агент не будет знать или задумываться об этом? Такой подход позволяет предложить очень простое решение: [*Left*]. Если мусор не был оставлен после прибытия агента в квадрат во время фактического выполнения плана, то агент обнаружит, что цель достигнута. В противном случае, поскольку предусловие *CleanL* неявного шага *Finish* не выполнено, агент выработает новый план: [*Suck*]. Выполнение этого плана всегда приводит к успеху.

Контроль выполнения и перепланирование, вместе взятые, образуют общую стратегию, которая может применяться и в полностью наблюдаемых, и в частично наблюдаемых вариантах среды, а также может распространяться на самые различные представления, применяемые в планировании, включая планы в пространстве состояний, планы с частичным упорядочением и условные планы. Один из простых подходов к планированию в пространстве состояний показан в листинге 12.5. Планирующий агент начинает с цели и создает начальный план ее достижения. Затем агент приступает к последовательному выполнению действий. А перепланирующий агент, в отличие от других рассматриваемых нами планирующих агентов, следит за выполнением как оставшегося невыполненного сегмента плана *plan*, так и полного первоначального плана *whole\_plan*. В нем используется ~~контроль~~ **контроль действий**: прежде чем выполнить следующее действие плана *plan*, агент проверяет свои результаты восприятия для определения того, не оказались ли вдруг невыполненными какие-либо предусловия плана. Если они действительно являются таковыми, то агент пытается снова попасть в колею, перепланируя последовательность действий, которые должны помочь ему вернуться в некоторый пункт плана *whole\_plan*.

**Листинг 12.5. Алгоритм агента, который занимается контролем действий и перепланированием.** В нем в качестве процедуры используется полный алгоритм планирования в пространстве состояний, называемый **Planner**. Если предусловия следующего действия не выполняются, агент проходит по циклу через возможные пункты *p* в плане *whole\_plan*, пытаясь найти такой из них, путь к которому может быть запланирован с помощью алгоритма **Planner**. Этот путь называется **repair** (поправка). Если в алгоритме **Planner** удается найти такую поправку **repair**, агент соединяет **repair** и остаток плана после пункта *p*, чтобы создать новый план. Затем агент возвращается к первому этапу нового плана

---

```

function Replanning-Agent(percept) returns действие action
  static: KB, база знаний (включает описания действий)
    plan, план, первоначально представленный пустым списком []
    whole_plan, общий план, первоначально представленный
      пустым списком []
    goal, цель

    Tell(KB, Make-Percept-Sentence(percept, t))
    current  $\leftarrow$  State-Description(KB, t)
    if plan = [] then

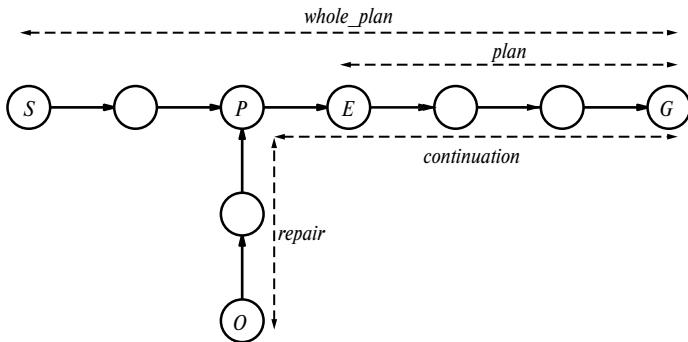
```

```

whole_plan ← plan ← Planner(current, goal, KB)
if предусловия Preconditions(First(plan)) не имеют в настоящее
время значения true in KB then
    candidates ← Sort(whole_plan, упорядоченный по расстояниям
                      до текущего состояния current)
    find состояние s среди возможных кандидатов candidates,
          такое, что failure ≠ repair ← Planner(current, s, KB)
    continuation ← хвост общего плана whole_plan, начинающийся с s
    whole_plan ← plan ← Append(repair, continuation)
return Pop(plan)

```

Схематически этот процесс приведен на рис. 12.9. Перепланирующий агент обнаруживает, что в текущем состоянии не выполнены предусловия первого действия в плане *plan*. Поэтому он вызывает алгоритм планирования, чтобы получить новый субплан *repair*, позволяющий перейти из текущего состояния в некоторое состояние *s* плана *whole\_plan*. В данном примере оказалось, что состояние *s* находится на один шаг назад от текущего, оставшегося плана *plan* (это удалось определить благодаря тому, что осуществляется контроль за выполнением всего плана, а не только оставшегося). Вообще говоря, следует выбирать состояние *s* так, чтобы оно было как можно ближе к текущему состоянию. Соединение плана *repair* и части плана *whole\_plan* от *s* и дальше, которую мы называем *continuation*, позволяет получить новый план *plan*, и агент становится готовым возобновить выполнение плана.



*Рис. 12.9. Перед выполнением планировщик предлагает план, называемый здесь whole\_plan, для перехода из состояния S в состояние G. Агент выполняет этот план до тех пор, пока не встретится пункт E. Прежде чем выполнить оставшийся план plan, агент, как обычно, проверяет предусловия и обнаруживает, что он фактически находится в состоянии O, а не в состоянии E. Затем он вызывает свой алгоритм планирования для получения поправки repair, которая представляет собой план перехода из пункта O в некоторый пункт P первоначального плана whole\_plan. Теперь новый план plan становится конкатнацией планов repair и continuation (продолжения первоначального плана whole\_plan).*

Теперь вернемся к примеру задачи, в которой нужно было добиться, чтобы цвет стула и стола совпадал, но на этот раз применим перепланирование. Предполагается

наличие полностью наблюдаемой среды. В начальном состоянии стул окрашен синей краской, стол — зеленой, и имеются банка с синей краской и банка с красной краской. Таким образом, может быть сформулировано следующее определение задачи:

```

Init(Color(Chair, Blue) ∧ Color(Table, Green)
     ∧ ContainsColor(BC, Blue) ∧ PaintCan(BC))
     ∧ ContainsColor(RC, Red) ∧ PaintCan(RC))
Goal(Color(Chair, x) ∧ Color(Table, x))

Action(Paint(object, color),
       Precond: HavePaint(color),
       Effect: Color(object, color))
Action(Open(can),
       Precond: PaintCan(can) ∧ ContainsColor(can, color),
       Effect: HavePaint(color))

```

Алгоритм Planner агента должен предложить такой план:

```
[Start; Open(BC); Paint(Table, Blue); Finish]
```

Теперь агент готов выполнить этот план. Предположим, что все шло хорошо, поскольку агент успешно открыл банку с синей краской и нанес эту краску на стол. В этот момент агенты, описанные в предыдущих разделах, объявили бы о победе, поскольку они завершили все этапы в плане. Но агент, контролирующий выполнение, должен вначале проверить предусловие этапа *Finish*, в котором сказано, что эти два предмета мебели должны иметь одинаковый цвет. Предположим, агент обнаружил, что стол и стул не имеют одинакового цвета, поскольку он не закрасил часть стола и осталось зеленое пятно. Теперь агент должен определить, в какую позицию плана *whole\_plan* он должен перейти и какая последовательность корректирующих действий позволяет в нее попасть. Агент обнаруживает, что текущее состояние идентично предусловию перед выполнением действия *Paint*, поэтому агент выбирает пустую последовательность для *repair* и принимает, что его план *plan* должен быть таким же, как и только что выполненная последовательность [*Paint*, *Finish*]. После принятия этого нового плана контроль выполнения возобновляется и предпринимается попытка выполнить действие *Paint*. Такое поведение повторяется в цикле до тех пор, пока результаты восприятия не будут указывать, что стол полностью перекрашен. Но заслуживает внимания то, что этот цикл создается процессом “запланировать–выполнить–перепланировать”, а не явным циклом в плане.

Контроль действий — это очень простой метод контроля выполнения, но он может иногда приводить к поведению, которое нельзя назвать интеллектуальным. Например, предположим, что агент формирует план решения задачи по перекраске, предусмотрев окрашивание стула и стола в красный цвет. Затем он открывает банку с красной краской и обнаруживает, что краски достаточно только для стула. При контроле действий неудача не была бы обнаружена до тех пор, пока не будет окрашен стул, поскольку в этот момент предусловие *HavePaint(Red)* становится ложным. А фактически требуется обнаруживать неудачное завершение каждый раз, когда состояние является таковым, что оставшийся план больше не будет работать. В методе **контроля плана** эта цель достигается путем проверки предусловий успеха всего оставшегося плана, т.е. предусловий каждого этапа в плане, за исключением

тех предусловий, которые достигаются с помощью еще одного этапа в оставшемся плане. В условиях контроля плана выполнение поставленного под сомнение плана прекращается настолько быстро, насколько это возможно, а не продолжается до тех пор, пока фактически не возникнет неудачное завершение<sup>13</sup>. В некоторых случаях такое поведение позволяет спасти агента от несчастья, когда поставленный под сомнение план завел бы его к тупику, из которого цель будет недостижима.

Модифицировать алгоритм планирования таким образом, чтобы в нем в каждом пункте план сверялся с предусловиями успеха оставшегося плана, относительно несложно. Если же мы расширим контроль плана для проверки того, выполняются ли в текущем состоянии предусловия плана в любом будущем пункте, а не только в текущем пункте, то контроль плана позволит также воспользоваться преимуществом **удачи**, т.е. случайного успеха. Если кто-то вдруг придет на помощь и покрасит стол в красный цвет одновременно с тем, как агент красит в красный цвет стул, то предусловия завершения плана будут выполнены (поскольку цель достигнута) и агент сможет раньше уйти с работы.

До сих пор методы контроля и перепланирования рассматривались в полностью наблюдаемых вариантах среды. А если среда является частично наблюдаемой, то ситуация может стать гораздо более сложной. Во-первых, обстоятельства могут так измениться в худшую сторону, что агент не сумеет этого обнаружить. Во-вторых, для “проверки предусловий” может потребоваться выполнение действий по получению информации с помощью датчиков, которые должны быть запланированы (либо на этапе планирования, что потребует от нас возврата к условному планированию, либо на этапе выполнения плана). В наихудшем случае для осуществления любого действия по получению информации с помощью датчиков может потребоваться составление сложного плана, который сам требует контроля и поэтому применения дополнительных действий по сбору информации, и т.д. Если агент упорно добивается того, чтобы были проверены все предусловия, то может так и не перейти к фактическому выполнению каких-либо действий. Агент должен предпочесть в такой ситуации проверку лишь переменных, которые имеют важное значение, характеризуются значительной вероятностью того, что их значения окажутся неправильными, и не требуют слишком больших затрат на сбор информации об их значениях. Это позволяет агенту отвечать должным образом на важные опасности, но не тратить время на проверку того, не падает ли небо на голову.

Теперь, после описания одного из методов контроля и перепланирования, мы должны ответить на вопрос: “Будет ли он работать?” Этот вопрос — на удивление сложный. Если под ним подразумевается: “Можем ли мы гарантировать, что агент всегда достигнет цели, даже в условиях неограниченной недетерминированности?”, то ответ будет отрицательным, поскольку агент может неожиданно попасть в тупиковую ситуацию, как описано применительно к оперативному поиску в разделе 4.5. Например, агент-пылесос может не знать, что его аккумуляторы способны разрядиться. Но мы исключим из рассмотрения тупиковые ситуации, т.е. предположим, что агент может сформировать план достижения цели из любого состояния в своей среде. Если принять предположение, что среда действительно является недетерми-

---

<sup>13</sup> Благодаря использованию контроля плана наш агент становится немного умнее, чем навозный жук (см. с. 81). Наш агент заметил бы, что шарик навоза в его лапках отсутствует, и поэтому перепланировал бы свои действия, чтобы получить еще один шарик навоза и заткнуть им свою норку.

нированной, в том смысле, что подобный план всегда имеет какой-то шанс на успех в любой конкретной попытке его выполнения, то агент в конечном итоге достигнет цели. Поэтому перепланирующий агент обладает способностями, аналогичными тем, которыми обладает агент, занимающийся условным планированием. В действительности можно модифицировать условный планировщик таким образом, чтобы он формировал только частичный план решения, который включает этапы в форме “**if** <test> **then** plan<sub>A</sub> **else** replan”. С учетом принятых выше предположений такой план может стать правильным решением первоначальной задачи, к тому же его формирование может оказаться намного дешевле по сравнению с созданием полного условного плана.

Неприятности возникают, когда повторные попытки достижения цели агентом становятся бесплодными — когда они блокируются каким-то предусловием или результатом, о котором агент не знает. Например, если агенту выдали неправильную магнитную карточку-ключ от его номера в гостинице, то он не сможет открыть дверь, даже вставляя эту карточку и вынимая бесконечное количество раз<sup>14</sup>. Одно из решений состоит в том, чтобы выбрать случайным образом один из множества возможных планов исправления ситуации, а не пытаться каждый раз выполнить один и тот же план. В данном случае полезной альтернативой был бы план исправления ситуации, состоящий в том, чтобы снова подойти к администратору гостиницы и получить правильную карточку-ключ от комнаты. Учитывая то, что агент может оказаться неспособным отличать друг от друга действительно недетерминированный случай и случай, в котором все попытки будут напрасными, такой способ организации выполнения плана, в котором предусмотрено использование нескольких вариантов исправления ситуации, в целом становится более целесообразным.

Еще одним решением проблемы неправильных описаний действий является **обучение**. После нескольких попыток обучающийся агент должен быть способен модифицировать описание действия, в котором указано, что данный ключ открывает дверь. С этого момента перепланировщик должен автоматически выработать альтернативный план, который состоит в получении нового ключа. Такого рода обучение рассматривается в главе 21.

Даже несмотря на все эти потенциальные усовершенствования, перепланирующий агент все еще обладает некоторыми недостатками. Он не может действовать в тех вариантах среды, где все происходит в реальном времени, и не существует пределов количества времени, которое он может затратить на перепланирование, и поэтому нет пределов времени, в течение которого он решится на выполнение какого-то действия. Кроме того, данный агент не способен формулировать новые цели самостоятельно или принимать к исполнению новые цели, дополнительно к его текущим целям, поэтому такой агент не сможет долго существовать в сложной среде. Указанные недостатки будут устранены в следующем разделе.

---

<sup>14</sup> Бессмысленное повторение одного и того же способа исправления плана полностью повторяет поведение, которое демонстрирует оса-сфекс (см. с. 81).

## 12.6. НЕПРЕРЫВНОЕ ПЛАНИРОВАНИЕ

В этом разделе рассматривается проект агента, способного существовать в некоторой среде неопределенно долго. Поэтому он не является “решателем задач”, который получает единственную цель, а затем составляет план и действует до тех пор, пока эта цель не будет достигнута; скорее, данный агент существует, проходя через целые ряды этапов формулировки постоянно меняющихся целей, планирования и осуществления действий. Вместо трактовки процессов планирования и контроля выполнения как отдельных процессов, один из которых передает свои результаты другому, их можно рассматривать как единый процесс, осуществляемый **непрерывно планирующим агентом** (*continuous planning agent*).

Такого агента можно рассматривать как находящегося постоянно в ситуации осуществления плана — грандиозного плана прожития его жизни. Его деятельность включает выполнение некоторых этапов плана, готовых к выполнению, уточнение плана для удовлетворения открытых предусловий или разрешения конфликтов, а также модификацию плана в свете дополнительной информации, полученной во время его выполнения. Очевидно, что агент, впервые приступая к формулировке новой цели, не будет иметь в своем распоряжении действий, готовых для выполнения, поэтому он потратит некоторое время на выработку частичного плана. Однако вполне возможно, что агент начнет выполнение плана еще до того, как разработка плана будет полностью закончена, особенно если в нем имеются независимые подцели, которых требуется достичь уже сейчас. Непрерывно планирующий агент постоянно контролирует состояние мира, обновляя свою модель мира по новым результатам восприятия, даже если его размышления над планом еще продолжаются.

Вначале рассмотрим один пример, а затем опишем программу агента, которая получила название *Continuous-POP-Agent*, поскольку в ней для представления намеченной деятельности используются планы с частичным упорядочением. Чтобы упростить это представление, мы будем предполагать наличие полностью наблюдаемой среды. Те же самые методы могут быть распространены и на частично наблюдаемый случай.

В качестве примера рассматривается задача из проблемной области мира блоков (раздел 11.1). Начальное состояние показано на рис. 12.10, *a*. Нам потребуется действие *Move(x, y)*, в котором блок *x* перемещается на блок *y*, при условии, что обе их верхние поверхности являются свободными. Схема этого действия состоит в следующем:

```
Action(Move(x, y))  
  Precond: Clear(x)  $\wedge$  Clear(y)  $\wedge$  On(x, z),  
  Effect: On(x, y)  $\wedge$  Clear(z)  $\wedge$   $\neg$ On(x, z)  $\wedge$   $\neg$ Clear(y)
```

Агенту вначале необходимо сформулировать для себя цель. Здесь мы не будем обсуждать процесс формулировки цели, а вместо этого предположим, что агенту было дано задание достичь цели *On(C, D)  $\wedge$  On(D, B)* (или он принял это решение самостоятельно). Агент начинает планировать действие по достижению данной цели. В отличие от всех других описанных в этой книге агентов, которые отключают все свои восприятия до тех пор, пока планировщик не вернет полное решение данной задачи, непрерывно планирующий агент формирует свой план поэтапно, затрачивая на каждый этап ограниченное количество времени. После каждого этапа планирования агент возвращает в качестве своего действия пустую операцию *NoOp* и снова

проверяет результаты своего восприятия. Предположим, что результаты восприятия не меняются и агент быстро составляет план, показанный на рис. 12.11. Обратите внимание на то, что предусловия обоих действий,  $Move(D, B)$  и  $Move(C, D)$ , выполнены еще в состоянии  $Start$ , но в соответствии с существующим ограничением упорядочения первое из них вносится в план прежде, чем второе. Это требуется для обеспечения того, чтобы предикат  $Clear(D)$  оставался истинным до выполнения действия  $Move(D, B)$ . На протяжении всего процесса непрерывного планирования в качестве метки для текущего состояния всегда используется  $Start$ . Агент обновляет это состояние после каждого действия.

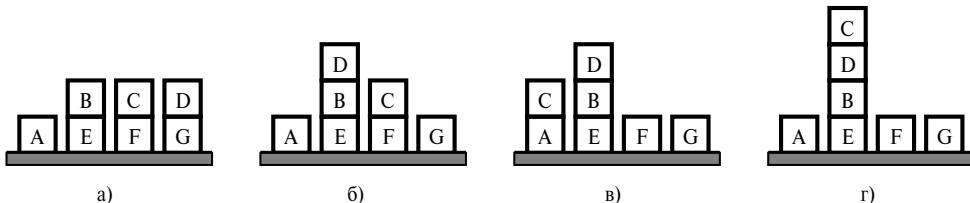


Рис. 12.10. Последовательность состояний, возникающих в ходе того, как непрерывно планирующий агент пытается достичь целевого состояния  $On(C, D) \wedge On(D, B)$ , как показано на рис. 12.10, г: начальное состояние (а); вмешательство другого агента, поставившего блок D на блок B (б); агент выполнял действие  $Move(C, D)$ , но допустил ошибку и вместо перемещения блока C на блок D уронил блок C на блок A (в); агент предпринял повторную попытку выполнить действие  $Move(C, D)$  и достиг целевого состояния (г)

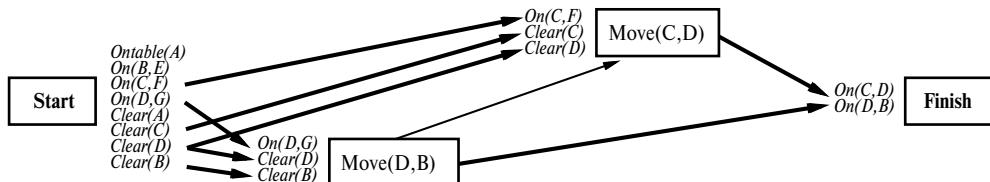


Рис. 12.11. Начальный план, сформированный непрерывно планирующим агентом. Пока что этот план нельзя отличить от плана, подготовленного обычным планировщиком с частичным упорядочением

Теперь план готов к выполнению, но прежде чем агент сможет приступить к действию, вмешивается природа. Какой-то внешний агент (возможно, учитель агента, потерявший терпение) перемещает блок D на блок B, после чего мир переходит в состояние, показанное на рис. 12.10, в. Агент воспринимает информацию об этом состоянии, учитывает, что предикаты  $Clear(B)$  и  $On(D, G)$  больше не являются истинными в текущем состоянии, и обновляет соответствующим образом модель текущего состояния. Причинные связи, которые создавали предусловия  $Clear(B)$  и  $On(D, G)$  для действия  $Move(D, B)$ , становятся недействительными и должны быть удалены из плана. Новый план показан на рис. 12.12. Состояние  $Start$  всегда представляет текущее состояние, но оно изменилось, поэтому на данном рисунке состояние  $Start$  отличается от соответствующего состояния на предыдущем рисунке. Обратите внимание на то, что теперь план стал неполным: два из предусловий для действия  $Move(D, B)$  являются открытыми, а его предусловие  $On(D, y)$  теперь неконкретизировано, поскольку больше нет смысла предполагать, что это перемещение будет выполнено из позиции блока G.

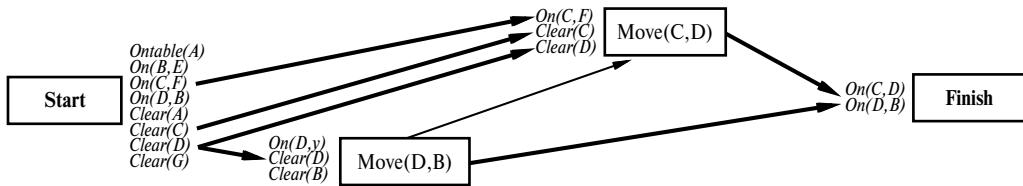


Рис. 12.12. После того как некто посторонний перемещает блок D на блок B, неподдерживаемые связи, создающие предусловия  $\text{Clear}(B)$  и  $\text{On}(D, G)$ , уничтожаются, в результате чего создается показанный здесь план

Теперь агент может воспользоваться преимуществом этого “полезного” вмешательства, обнаружив, что причинная связь  $\text{Move}(D, B) \xrightarrow{\text{On}(D, B)} \text{Finish}$  может быть заменена прямой связью от состояния *Start* к состоянию *Finish*. Этот процесс называется **продлением** (extension) причинной связи и осуществляется каждый раз, когда некоторое условие может быть взято из более раннего, а не более позднего этапа без создания нового конфликта.

После удаления старой причинной связи от  $\text{Move}(D, B)$  к *Finish* действие  $\text{Move}(D, B)$  больше не поддерживает вообще какие-либо причинные связи. Теперь оно становится **избыточным этапом** (redundant step). Все избыточные этапы и все поддерживающие их связи удаляются из плана. В результате создается план, показанный на рис. 12.13.

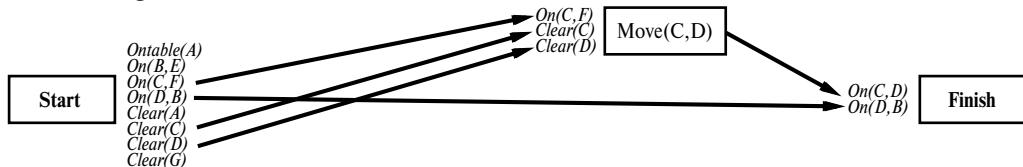


Рис. 12.13. Связь, поддерживаемая действием  $\text{Move}(D, B)$ , теперь заменена связью, проходящей от состояния *Start*, а этап  $\text{Move}(D, B)$ , ставший избыточным, удален

После этого стал готовым к выполнению этап  $\text{Move}(C, D)$ , поскольку все его предусловия выполнены на этапе *Start*, перед ним не требуется проходить какие-либо другие этапы и он не конфликтует с какой-либо другой связью в плане. Данный этап удаляется из плана и выполняется. К сожалению, агент допустил оплошность и уронил блок C на блок A, вместо того чтобы поставить его на D, поэтому было получено состояние, показанное на рис. 12.10, в. Новое состояние плана приведено на рис. 12.14. Обратите внимание на то, что в плане нет новых действий, но все еще остается открытое условие для этапа *Finish*.



Рис. 12.14. После того как действие  $\text{Move}(C, D)$  выполнено и удалено из плана, результаты этапа *Start* отражают тот факт, что в конечном итоге блок C оказался на блоке A, а не на блоке D, для установки на котором он предназначался. Предусловие цели  $\text{On}(C, D)$  все еще открыто

Агент решает составить план для этого открытого условия. Опять-таки целевое условие будет выполнено с помощью действия  $Move(C, D)$ . Его предусловия удовлетворяются за счет новых причинных связей от этапа  $Start$ . Новый план приведен на рис. 12.15.

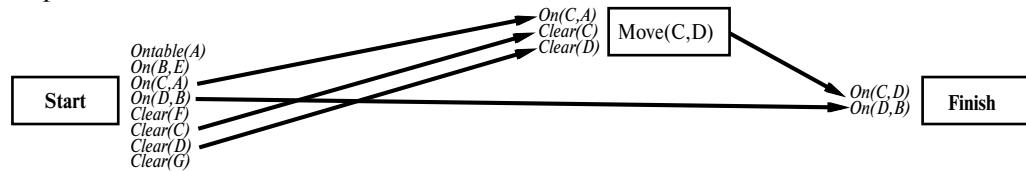


Рис. 12.15. Открытое предусловие удовлетворяется за счет повторного введения в план действия  $Move(C, D)$ . Обратите внимание на то, что для предусловий применяются новые связывания

Действие  $Move(C, D)$  снова готово к выполнению. На этот раз оно завершается успешно и достигается целевое состояние, показанное на рис. 12.10, г. После удаления данного этапа из плана целевое условие  $On(C, D)$  снова становится открытым. Но поскольку этап  $Start$  обновлен и в нем отражено новое состояние мира, целевое условие может быть немедленно выполнено с помощью связи от этапа  $Start$ . Таковым является обычный ход событий, когда какое-то действие оказывается успешным. Конечное состояние данного плана показано на рис. 12.16. Поскольку все целевые условия удовлетворяются этапом  $Start$  и больше нет оставшихся действий, агент теперь вправе удалить все подцели из состояния  $Finish$  и сформулировать новую цель.

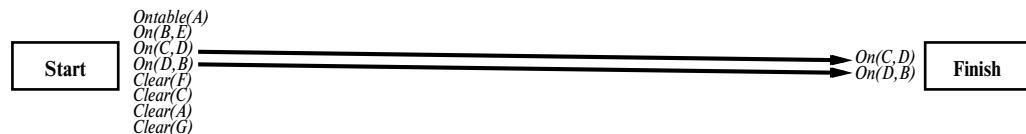


Рис. 12.16. После того как действие  $Move(C, D)$  выполняется и удаляется из плана, оставшееся открытое условие  $On(C, D)$  может быть разрешено путем добавления причинной связи от нового этапа  $Start$ . Теперь выполнение плана закончено

Из этого примера ясно, что непрерывное планирование весьма напоминает планирование с частичным упорядочением. В каждой итерации алгоритм находит что-то, касающееся плана, что требует исправления (так называемые **дефекты плана** — *plan flaw*), и исправляет это положение. Сам алгоритм POP также может рассматриваться как алгоритм устранения дефектов, в котором учитываются два дефекта: открытые предусловия и конфликты причинных связей. Непрерывно планирующий агент, с другой стороны, справляется с гораздо более широким перечнем дефектов, которые описаны ниже.

- Недостающие цели.** Агент может принять решение по добавлению в состояние  $Finish$  новой цели или целей. (При непрерывном планировании было бы больше смысла применять вместо названия  $Finish$  название *Infinity*, а вместо  $Start$  — *Current*, но авторы будут придерживаться установленвшейся традиции.)
- Открытые предусловия.** Агент добавляет причинную связь к открытому предусловию, выбирая либо новое, либо существующее действие (как в алгоритме POP).

- **Конфликты причинных связей.** При наличии причинной связи  $A \xrightarrow{p} B$  и действия  $C$  с результатом  $\neg p$  агент для разрешения этого конфликта выбирает ограничение упорядочения или ограничение переменной (как в алгоритме POP).
- **Неподдерживаемые связи.** Если есть причинная связь  $Start \xrightarrow{p} A$ , где предикат  $p$  больше не является истинным в состоянии  $Start$ , агент удаляет эту связь (что позволяет предотвратить выполнения действия, предусловия которого являются ложными.)
- **Избыточные действия.** Если какое-то действие  $A$  не обеспечивает выполнения ни одной причинной связи, агент удаляет и это действие, и его связи (что позволяет ему воспользоваться преимуществом удачно сложившихся обстоятельств.)
- **Невыполненные действия.** Если некоторое действие  $A$  (отличное от  $Finish$ ) имеет все выполненные предусловия в состоянии  $Start$ , но связано с упорядоченными перед ним другими действиями (кроме  $Start$ ) и не конфликтует ни с одной причинной связью, агент удаляет из плана действие  $A$  вместе с его причинными связями и возвращает его как действие, которое должно быть выполнено.
- **Ненужные, существовавшие в прошлом цели.** Если в плане нет открытых предусловий и нет действий (поэтому все причинные связи проходят непосредственно от состояния  $Start$  к состоянию  $Finish$ ), то текущее множество целей достигнуто. Агент удаляет эти цели и связи, ведущие к ним, чтобы освободить место для новых целей.

Алгоритм Continuous-POP-Agent приведен в листинге 12.6. В нем имеется цикл “принять результаты восприятия, удалить дефекты, выполнить действие”. Агент, действующий в соответствии с этим алгоритмом, хранит в своей базе знаний постоянно существующий план и в каждой операции удаляет из этого плана по одному дефекту. Затем он выполняет некоторое действие (хотя таким действием часто бывает пустая операция  $NoOp$ ) и повторяет цикл. Этот агент способен преодолевать многие проблемы, перечисленные при обсуждении перепланирующего агента на с. 599. В частности, он может действовать в реальном времени, пользуясь удачным стечением обстоятельств, способен формулировать цели самостоятельно и справляется с непредвиденными ситуациями, которые влияют на выполнение плана в будущем.

**Листинг 12.6. Алгоритм непрерывно планирующего агента с частичным упорядочением Continuous-POP-Agent.** После получения очередных результатов восприятия агент удаляет один дефект из своего постоянно обновляемого плана, а затем возвращает действие. Агенту часто приходится выполнять много этапов планирования, связанных с удалением дефектов, в течение которых он возвращает пустую операцию  $NoOp$ , пока он не будет готов выполнить какое-то реальное действие

---

```

function Continuous-POP-Agent(percept) returns действие action
  static: plan, план, первоначально содержащий только
           действия Start и Finish

  action  $\leftarrow$  NoOp (действие NoOp применяется по умолчанию)
  Effects[Start] = Update(Effects[Start], percept)
  Remove-Flaw(plan) // Возможно, выполнение этой операции приведет
                      // к обновлению действия action
  return action

```

---

## 12.7. МУЛЬТИАГЕНТНОЕ ПЛАНИРОВАНИЕ

До сих пор нам приходилось иметь дело только с **одноагентными вариантами среды**, в которых рассматриваемый агент действует в одиночестве. А если в этой среде есть также другие агенты, наш агент может просто включить их в свою модель среды, не изменяя своих основных алгоритмов. Но во многих случаях такой подход приводит к низкой производительности, поскольку взаимодействие с другими агентами во многом отличается от взаимодействия с природой. В частности, природа (как обычно принято считать) безразлична к намерениям агента<sup>15</sup>, а другие агенты — нет. В данном разделе приведены основные сведения о мультиагентном планировании, которое предназначено для решения указанных проблем.

Как было показано в главе 2, мультиагентные варианты среды могут быть **кооперативными** или **конкурентными**. Начнем с простого кооперативного примера: планирование действий команды в парном теннисе. Для определения действий обоих игроков в команде могут быть составлены планы; в этом разделе будут описаны методы эффективного формирования таких планов. Эффективно сформированный план полезен, но не гарантирует успеха; прежде всего агенты должны согласиться использовать один и тот же план! Для этого требуется определенная форма **координации**, которая может быть достигнута с помощью **общения**.

### Кооперация: совместные цели и планы

Два агента, играющие в одной команде в парный теннис, имеют единую цель — выиграть матч, что приводит к возникновению различных подцелей. Предположим, что в какой-то момент игры они имеют общую цель — отбить мяч, который направлен на их половину поля, и обеспечить, чтобы по меньшей мере один из них играл под сеткой. Мы можем представить это общее намерение в виде задачи **мультиагентного планирования**, как показано в листинге 12.7.

**Листинг 12.7.** Задача игры в парный теннис. Два агента играют в одной команде и могут присутствовать в одном из четырех мест: **[Left, Baseline]** (слева, у задней линии), **[Right, Baseline]** (справа, у задней линии), **[Left, Net]** (слева, под сеткой) и **[Right, Net]** (справа, под сеткой). Мяч может быть отбит, если в нужном месте находится один и только один игрок

---

```

Agents(A, B)
Init(At(A, [Left, Baseline]) ∧ At(B, [Right, Net])) ∧
    Approaching(Ball, [Right, Baseline])) ∧ Partner(A,B) ∧ Partner(B,A)
Goal(Returned(Ball) ∧ At(agent, [x, Net]))

Action(Hit(agent, Ball),
    Precond: Approaching(Ball, [x, y]) ∧ At(agent, [x, y]) ∧
        Partner(agent, partner) ∧ ¬At(partner, [x, y]),
    Effect: Returned(Ball))
Action(Go(agent, [x, y]),
    Precond: At(agent, [a, b]),
    Effect: At(agent, [x,y]) ∧ ¬At(agent, [a, b]))

```

---

В этой постановке задачи применяются два новых средства. Во-первых, в высказывании *Agents*(*A*, *B*) объявляется, что в плане участвуют два агента, *A* и *B* (по условиям данной задачи противостоящие им игроки не рассматриваются как агенты). Во-вторых, в каждом действии в качестве формального параметра упоминается агент, поскольку нам необходимо следить за тем, что делает каждый агент.

Решением мультиагентной задачи планирования является **совместный план** (joint plan), состоящий из действий для каждого агента. Совместный план представляет собой решение, если цель будет достигнута при условии, что каждый агент выполнит назначенные ему действия. Решением данной задачи игры в теннис является приведенный ниже план.

*Plan 1:*

```
A: [Go(A, [Right, Baseline]), Hit(A, Ball)]
B: [NoOp(B), NoOp(B)]
```

Если оба агента имеют одинаковую базу знаний и если это решение является единственным, то все сложится идеально; каждый из агентов сможет определить это решение, а затем совместно выполнить его. Но к большому сожалению для этих агентов (и мы вскоре увидим, почему им приходится об этом сожалеть), существует другой план, позволяющий так же успешно достичь цели, как и первый:

*Plan 2:*

```
A: [Go(A, [Left, Net]), NoOp(A)]
B: [Go(B, [Right, baseline]), Hit(B, Ball)]
```

Если агент *A* выберет план 2, а агент *B* — план 1, то никто из них не отобьет мяч. И наоборот, если агент *A* выберет план 1, а агент *B* — план 2, то они, вероятно, столкнутся друг с другом; ни один из них не отобьет мяч, и к тому же пространство под сеткой может остаться неприкрытым. Поэтому само существование правильных совместных планов еще не означает, что цель будет достигнута. Агентам нужен некоторый механизм **координации** для достижения одного и того же совместного плана; более того, оба агента должны обладать общими знаниями (см. главу 10) о том, что должен быть выполнен некоторый конкретный совместный план.

## Многотельное планирование

В данном разделе речь пойдет в основном о формировании правильных совместных планов, а вопросы координации мы пока отложим. Авторы называют данный подход **многотельным планированием** (multibody planning); по сути именно в этом состоит задача планирования, с которой сталкивается также один централизованный агент, который может раздавать указания по выполнению действий каждой из нескольких физических сущностей. А если рассматривается случай, в котором действует несколько настоящих агентов, такое планирование дает возможность каждому агенту выяснить, каковы возможные совместные планы, которые позволили бы агентам добиться успеха, если бы они действовали согласованно.

Применяемый нами подход к многотельному планированию будет основан на планировании с частичным упорядочением, которое описано в разделе 11.3. Чтобы упростить решение этой проблемы, мы будем исходить из предположения о полной

---

<sup>15</sup> С нами могут не согласиться жители Соединенного Королевства, где сами действия по подготовке пикника гарантируют дождь.

наблюдаемости среды. Есть еще один дополнительный вопрос, который не возникает в одноагентном случае: среда больше не является в полном смысле этого слова **статической**, поскольку другие агенты могут действовать, пока какой-то конкретный агент размышляет. Поэтому необходимо позаботиться о **синхронизации**. Для упрощения мы будем предполагать, что каждое действие занимает одно и то же количество времени и что действия, выполняемые в каждом пункте совместного плана, являются одновременными.

В любой момент времени каждый агент выполняет одно и только одно действие (возможно, включая пустую операцию *NoOp*). Такое множество одновременных действий называется **совместным действием** (joint action). Например, совместным действием в проблемной области тенниса (с. 606) с агентами *A* и *B* является  $\langle \text{NoOp}(A), \text{Hit}(B, \text{Ball}) \rangle$ . Совместный план состоит из частично упорядоченного графа совместных действий. Например, план 2 для описанной выше задачи игры в теннис может быть представлен как такая последовательность совместных действий:

```
<Go(A, [Left, Net]), Go(B, [Right, Baseline])>
<NoOp(A), Hit(B, Ball)>
```

Это планирование может быть выполнено с помощью обычного алгоритма POP, применяемого к множеству всех возможных совместных действий. Единственная проблема состоит в огромных размерах этого множества: при наличии 10 действий и 5 агентов количество совместных действий составляет  $10^5$ . Было бы очень утомительным правильное определение предусловий и результатов каждого действия, а разработка плана на таком большом множестве стала бы неэффективной.

Альтернативный подход заключается в том, что совместные действия определяются неявно путем описания того, как каждое отдельное действие сочетается с другими возможными действиями. Такой подход должен быть проще, поскольку большинство действий независимы друг от друга, поэтому нам потребуется перечислить лишь немного действий, которые действительно взаимодействуют друг с другом. Это можно сделать, дополнив обычные описания действий Strips или ADL одним новым средством — **списком одновременных действий** (concurrent action list). Этот список аналогичен предусловию любого описания действия, за исключением того, что в нем описываются не переменные состояния, а действия, которые должны или не должны выполняться одновременно. Например, действие удара по мячу, *Hit*, может быть описано следующим образом:

```
Action(Hit(A, Ball),
       Concurrent:  $\neg \text{Hit}(B, \text{Ball})$ ,
       Precond: Approaching(Ball, [x, y])  $\wedge$  At(A, [x, y]),
       Effect: Returned(Ball))
```

В данном случае мы встречаемся с ограничением, запрещающим одновременное выполнение, согласно которому при выполнении действия *Hit* одним агентом не должно быть выполнения действия *Hit* другим агентом. А иногда как раз и требуются одновременные действия, например, если портативный холодильник, заполненный напитками, смогут отнести на теннисный корт только два агента. В описании для этого действия должно быть сказано, что агент *A* не сможет выполнить действие *Carry*, если нет другого агента, *B*, который одновременно выполнил бы действие *Carry* применительно к тому же холодильнику:

```

Action(Carry(A, cooler, here, there),
       Concurrent: Carry(B, cooler, here, there),
       Precond: At(A, here) ∧ At(cooler, here) ∧ Cooler(cooler),
       Effect: At(A, there) ∧ At(cooler, there) ∧ ¬At(A, here) ∧
               ¬At(cooler, here))

```

При использовании такого представления появляется возможность создать планировщик, который весьма напоминает планировщик с частичным упорядочением, действующий по алгоритму РОР. Но между этими двумя планировщиками существуют три различия, которые описаны ниже.

1. Кроме отношения временного упорядочения  $A \prec B$  разрешается использовать отношения  $A = B$  и  $A \preceq B$ , которые означают “одновременно” и “прежде или одновременно”, соответственно.
2. Если какое-то новое действие требует одновременных действий, необходимо конкретизировать эти действия, используя новые или существующие действия в плане.
3. Запрещенные одновременные действия являются дополнительным источником ограничений. Каждое ограничение должно быть разрешено путем наложения на конфликтующие действия таких ограничений, чтобы они могли быть выполнены либо прежде, либо позже другого конфликтующего действия.

Такое представление позволяет создать алгоритм планирования для многотельных проблемных областей, эквивалентный алгоритму РОР. Мы могли бы дополнить этот подход с помощью уточнений, приведенных в последних двух главах (сетей HTN, частичной наблюдаемости, условных планов, контроля выполнения и перепланирования), но такая задача выходит за рамки настоящей книги.

## Механизмы координации

Простейший метод, с помощью которого группа агентов может обеспечить координацию при выполнении совместного плана, состоит в принятии определенного **соглашения** (convention) до начала совместной деятельности. Соглашением является любое ограничение, касающееся выбора совместных планов, выходящее за рамки того основного ограничения, в соответствии с которым совместный план должен работать, если ему следуют все агенты. Например, соглашение “придерживаться своей стороны поля” станет причиной того, что партнеры в парном теннисе выберут план 2, а соглашение, что “один игрок всегда остается у сетки”, приведет их к плану 1. Некоторые соглашения, такие как вождение по правильной стороне дороги, приняты настолько широко, что считаются **общественными законами**. Естественные языки также могут рассматриваться как соглашения.

Соглашения, указанные в предыдущем абзаце, являются характерными для конкретной проблемной области и могут быть реализованы путем внесения в описания действий таких ограничений, которые позволили бы исключить нарушения соответствующего соглашения. Более общий подход состоит в использовании соглашений, независимых от проблемной области. Например, если каждый агент эксплуатирует один и тот же алгоритм многотельного планирования с одними и теми же входными данными, то может следовать соглашению о выполнении первого найденного осу-

ществимого совместного плана и быть уверенным в том, что другие агенты сделают такой же выбор. Более надежная, но и более дорогостоящая стратегия могла бы состоять в том, чтобы выработать все совместные планы, а затем выбрать, например, тот из них, внешнее представление для вывода на печать которого находится на первом месте в алфавитном порядке.

Соглашения могут также возникать благодаря эволюционным процессам. Например, колонии общественных насекомых выполняют очень сложные совместные планы, а осуществление подобных действий к тому же обеспечивается благодаря общим генетическим характеристикам отдельных особей в этой колонии. Согласованность действий может также быть обусловлена тем фактом, что отход от соглашений приводит к уменьшению эволюционной приспособляемости всех особей, поэтому любой осуществимый совместный план может стать шагом к стабильному равновесию. Аналогичные условия распространяются и на процесс развития любого человеческого языка, когда важным становится не тот факт, на каком языке должен говорить каждый индивидуум, а тот факт, что все индивидуумы должны говорить на одном и тот же языке. Наш последний пример относится к тому, как ведут себя птицы, сбиваясь в стаи во время полета. Вполне приемлемая модель такого процесса может быть получена, если каждый птицеподобный агент (который иногда именуется *орнитоидом* или *бoidом* от слова *boid*, или *birdoid*) выполняет три перечисленных ниже правила, применяя определенный метод их комбинирования.

- Отделение.** Направлять свой полет подальше от соседей, если ты начинаешь к ним слишком заметно приближаться.
- Соединение в линию.** Придерживаться направления полета, позволяющего занять среднюю позицию по отношению к соседям.
- Выравнивание.** Рулить в сторону средней ориентации (направления движения) соседей.

Если все птицы выполняют одни и те же описанные выше правила, то вся стая обнаруживает *эмерджентное поведение* (от слова *emergent* — появляющийся), совершая полет в виде одного псевдоустойчивого тела приблизительно с постоянной плотностью, которое не рассеивается со временем. Как и в случае общественных насекомых, для каждого агента не требуется обладание совместным планом, который моделировал бы действия других агентов.

Как правило, принимаются такие соглашения, которые охватывают целый универсум отдельных задач мультиагентного планирования, а не требуют разработки с нуля перед началом решения каждой новой задачи. Это может приводить к недостаточной гибкости и нарушениям в работе, как иногда можно наблюдать в парном теннисе, когда мяч пролетает приблизительно на равных расстояниях между двумя партнерами. В отсутствие применимого соглашения агенты могут использовать **общение** для получения общих знаний об осуществимом совместном плане. Например, в парном теннисе игрок может крикнуть “Мой!” или “Твой！”, имея в виду мяч, чтобы указать на предпочтительный для него совместный план. Механизмы общения рассматриваются более подробно в главе 22, в которой отмечается, что общение не обязательно требует устного обмена репликами. Например, один игрок может неявно сообщить о предпочтительном совместном плане другому, просто выполнив его первую часть. В нашей задаче игры в теннис, если агент A направился к сетке, то

агент  $B$  обязан отойти назад, к линии подачи, чтобы отбить мяч, поскольку план 2 является единственным совместным планом, который начинается с того, что агент  $A$  направляется к сетке. Такой подход к координированию действия, иногда называемый **распознаванием плана** (plan recognition), является применимым, если для безошибочного определения нужного совместного плана достаточно одного действия (или краткой последовательности действий).

Вся нагрузка по определению того, как агенты должны приходить к успешному совместному плану, может быть возложена либо на проектировщиков агентов, либо на самих агентов. В первом случае проектировщик агента должен доказать, что правила и стратегии агентов будут успешными, еще до того, как агенты приступят к планированию. Сами агенты смогут правильно и успешно реагировать на обстановку, если заложенные в них правила и стратегии применимы для той среды, в которой они существуют, и им не нужно иметь явные модели, описывающие действия других агентов. А в последнем случае агентам придется больше рассуждать; они должны доказывать или демонстрировать другими способами, что их планы будут успешными, принимая во внимание рассуждения других агентов. Например, в среде с двумя логическими агентами,  $A$  и  $B$ , оба этих агента могут иметь следующее определение:

$$\forall p, s \text{ Feasible}(p, s) \Leftrightarrow \text{CommonKnowledge}(\{A, B\}, \text{Achieves}(p, s, \text{Goal}))$$

В нем указано, что в любой ситуации  $s$  план  $p$  представляет собой осуществимый совместный план в данной ситуации, если оба агента располагают знаниями о том, что план  $p$  позволяет достичь цели. Нам потребуются дополнительные аксиомы для формирования общих знаний о **совместном намерении** (joint intention) выполнить конкретный совместный план; только после этого агенты могут приступать к действиям.

## Конкуренция

Не все мультиагентные варианты среды включают кооперирующих агентов. Агенты с конфликтующими функциями полезности находятся в состоянии **конкуренции** друг с другом. Одним из примеров этого являются игры между двумя игроками с нулевой суммой, такие как шахматы. Как было описано в главе 6, агенту, играющему в шахматы, приходится рассматривать возможные ходы противника на несколько этапов в будущее. Это означает, что любой агент в конкурентной среде должен, во-первых, признавать наличие других агентов, во-вторых, прогнозировать некоторые из возможных планов другого агента, в-третьих, определять наилучшее действие с учетом указанных влияний. Поэтому в условиях конкуренции, как и в условиях кооперации, требуется модель с описанием планов другого агента. С другой стороны, в конкурентной среде действия агентов не вносят свой вклад в совместный план.

В разделе 12.4 проводится аналогия между играми и задачами условного планирования. Алгоритм условного планирования, приведенный в листинге 12.4, позволяет составлять планы, действующие при наихудших предположениях о данной среде, поэтому эти планы могут применяться в таких конкурентных ситуациях, когда агента интересует только успех или неудача. А если для агента и его противников важна также стоимость плана, то применимым становится алгоритм минимакса. До сих пор еще очень мало сделано в области совместного использования алгоритма

минимакса с такими методами, как планирование POP и HTN, которые выходят за рамки модели поиска в пространстве состояний, применяемой в главе 6. Мы вернемся к вопросу конкуренции в разделе 17.6, где рассматривается теория игр.

## 12.8. РЕЗЮМЕ

В данной главе описаны некоторые сложности, связанные с планированием и осуществлением действий в реальном мире. Основные идеи, изложенные в этой главе, перечислены ниже.

- Во многих действиях потребляются **ресурсы**, такие как деньги, топливо или сырье. Эти ресурсы удобно в целом рассматривать как числовые величины, а не пытаться рассуждать, скажем, о каждой отдельной монете или купюре во всем мире. Действия способны вырабатывать и потреблять ресурсы, поэтому обычно дешевле и эффективнее проверять частичные планы на предмет удовлетворения в них ресурсных ограничений, прежде чем предпринимать попытки их дальнейшего уточнения.
- Время — это один из наиболее важных ресурсов. За его расходованием можно следить, применяя специализированные алгоритмы составления расписаний или объединяя составление расписаний с планированием.
- Планирование с помощью **иерархической сети задач** (Hierarchical Task Network — HTN) позволяет агенту получить совет от проектировщика проблемной области в форме правил декомпозиции. Такой подход обеспечивает создание очень больших планов, требуемых для многих реальных приложений.
- В стандартных алгоритмах планирования предполагается наличие полной и правильной информации, а также детерминированной, полностью наблюдаемой среды. Это предположение является недействительным во многих проблемных областях.
- С проблемой неполной информации в планировании можно справиться, используя действия по применению датчиков для получения необходимой информации. **Условные планы** позволяют агенту получать с помощью датчиков информацию о мире во время выполнения плана для определения того, по какой ветви плана он должен следовать дальше. В некоторых случаях может использоваться **планирование без использования датчиков** или **согласованное планирование** для формирования плана, который во время своего выполнения не требует применения результатов восприятия. И планы без использования датчиков, и условные планы могут быть сформированы по методу поиска в пространстве **доверительных состояний**.
- Неправильная информация приводит к тому, что предусловия действий и планов остаются невыполненными. **Контроль выполнения** позволяет обнаруживать нарушения предусловий, создавая предпосылки успешного осуществления плана.
- В **перепланирующем агенте** используется контроль выполнения и по мере необходимости осуществляются восстановительные действия для возврата к первоначальному плану.

- **Непрерывно планирующий** агент создает новые цели в процессе своей деятельности и реагирует на изменения ситуации в реальном времени.
- **Мультиагентное** планирование необходимо, если в среде имеются другие агенты, с которыми приходится кооперировать, конкурировать или координировать свои действия. В **многотельном** планировании формируются совместные планы с использованием эффективной декомпозиции описаний совместных действий, но это планирование должно дополняться определенной формой координации, если два кооперативных агента должны согласовать друг с другом, какой из совместных планов следует выполнить.

---

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

---

Планирование с непрерывным учетом времени было впервые реализовано в программе Deviser [1541]. Проблема систематического представления времени в планах была решена Дином и др. [358] в системе Forbin. Программы Nonlin+ [1496] и Sipe [1593], [1594] обладают способностью формировать рассуждения о распределении ограниченных ресурсов по различным этапам плана. Программа O-Plan [93] (планировщик HTN) поддерживает равномерное, общее представление для ограничений, распространяющихся на время и ресурсы. Кроме приложения для Hitachi, упомянутого в этой главе, программа O-Plan применялась для планирования поставок программного обеспечения в компании Price Waterhouse и для планирования сборки заднего моста автомобиля в компании Jaguar Cars. Кроме того, был разработан целый ряд гибридных систем планирования и составления расписаний: система Isis [489], [490] использовалась при составлении производственных расписаний компании Westinghouse, система Gari [392] осуществляла планирование машинной обработки и конструирования механических деталей, система Forbin применялась для управления фабрикой, а система Nonlin+ служила средством планирования поставок для военно-морского флота.

После первоначального стремительного наплыва теоретических работ в области временного планирования в конце 1980-х годов наступило затишье, и лишь недавно интерес к этой теме возобновился в связи с тем, что появились новые алгоритмы и возросли обрабатывающие мощности, что привело к появлению возможности создавать новые практические приложения. В двух разработанных недавно планировщиках, Sapa [401] и T4 [627], используется прямой поиск в пространстве состояний в сочетании со сложными эвристическими функциями для учета действий, характеризующихся различными продолжительностями и требующих разных ресурсов. Альтернативой этого подхода является применение очень выразительных языков действий, но поиск в подобных системах должен осуществляться под управлением составленных людьми эвристик, характерных для данной проблемной области, как это сделано в системах ASPEN [511], HSTS [744] и IxTeT [549].

Разработки в области составления расписаний для аэрокосмических проектов имеют долгую историю. Программа T-Sched [412] использовалась для составления расписаний, представляющих собой последовательности принципиально важных команд для спутника Uosat-II. Программы Optimum-AIV [1] и Plan-ERS1 [509], основанные на программе O-Plan, применялись в Европейском космическом агентст-

ве, соответственно, для сборки космических аппаратов и планирования наблюдений. Программа Spike [741] использовалась для планирования наблюдений с помощью космического телескопа Хаббл в NASA, а система Space Shuttle Ground Processing Scheduling System [355] осуществляла составление производственного расписания с охватом вплоть до 16 000 рабочих смен. Программа Remote Agent [1108] стала первой автономной программой планирования и составления расписаний, применяемой для управления космическим аппаратом, которая работала на борту космического аппарата Deep Space One в 1999 году. В [1527] приведен обзор литературы по применению средств составления производственных расписаний в области исследования операций; теоретические результаты приведены в [993].

Применяемые в программе Strips средства для изучения **макроопераций** (“микрооператоров”, состоящих из последовательности примитивных этапов) могут рассматриваться как первый механизм иерархического планирования [465]. Иерархия использовалась также в системе Lawaly [1410]. В системе Abstrips [1336] была реализована идея **иерархии абстракции**, на основе которой было разрешено игнорировать предусловия действий низкого уровня при планировании на более высоких уровнях, чтобы можно было проще выявить общую структуру рабочего плана. В тезисах докторской диссертации Остина Тэйта [1494] и в работе Эрла Сакердоти [1338] были разработаны основные идеи планирования HTN в его современной форме. Многие практически применяемые планировщики, включая O-Plan и Sipe, представляют собой планировщики HTN. В [1628] обсуждаются свойства действий, благодаря которым планирование HTN становится эффективным. В [443], [444] представлен планировщик с полной иерархической декомпозицией, а также приведен ряд результатов анализа сложности для чистых планировщиков HTN. Другие авторы [25], [72], [766], [1635] предложили гибридный подход, принятый в данной главе, согласно которому декомпозиции просто представляют собой еще одну форму уточнения, которая может использоваться в планировании с частичным упорядочением.

Начиная с исследований по использованию микрооператоров в языке Strips, одной из целей иерархического планирования было повторное использование ранее накопленного опыта планирования в форме обобщенных планов. В некоторых системах, включая Soar [881] и Prodigy [221], в качестве средства обобщения ранее вычисленных планов применялся метод **обучения на основе объяснений** (explanation-based learning), описанный более подробно в главе 19. Альтернативный подход состоит в том, что ранее вычисленные планы должны сохраняться в их первоначальной форме, а затем использоваться повторно для решения новых подобных проблем по аналогии с первоначальной проблемой. Именно этот подход принят в области, получившей название **планирование на основе precedентов** (case-based planning) [23], [220], [609]. В [765] приведено обоснование того мнения, что планирование на основе precedентов должно рассматриваться как форма планирования на основе уточнения и что этот метод может служить средством формального описания для планирования с частичным упорядочением на основе precedентов.

На самых ранних этапах осуществления робототехнических проектов, в которых использовались методы планирования, включая Shakey [465] и Freddy [1045], была обнаружена проблема непредсказуемости и частичной наблюдаемости реальных вариантов среды. Эта проблема стала привлекать больше внимания после публикации статьи Макдермотта *Planning and Acting* [1021], оказавшей значительное влияние на ход дальнейших исследований.

В ранних планировщиках, в которых отсутствовали условные выражения и циклы, не была явно реализована концепция условного планирования, но несмотря на это, некоторые из таких планировщиков в ответ на неопределенность среды иногда прибегали к стилю, предусматривающему принудительный переход в заданное состояние. В программе Noah, разработанной Сакердоти, такой принудительный переход использовался в предлагаемом программой решении задачи с “ключами и ящиками”. Это — известная задача в области планирования, в которой планировщик имеет мало информации о начальном состоянии. Мэйсон [996] доказал, что в робототехническом планировании часто можно и нужно отказываться от сбора информации с помощью датчиков, и описал план без использования датчиков, позволяющий передвинуть инструмент в заданную позицию на столе с помощью последовательности раскачивающих действий, независимой от начальной позиции. Мы опишем эту идею в контексте робототехники (рис. 25.16).

Голдмен и Бодди [572] ввели термин **согласованное планирование** (*conformant planning*) применительно к планировщикам без использования датчиков, которые позволяют справиться с неопределенностью, принудительно переводя мир в известные состояния, и отметили, что планы без использования датчиков часто эффективнее, даже если агент имеет датчики. Первым достаточно эффективным согласованным планировщиком была программа Conformant Graphplan, или CGP, Смита и Уэлда [1437]. Феррарис и Гуинкилья [464], а также Рингтанен [1289] независимо разработали согласованные планировщики на основе алгоритма SAT<sub>plan</sub>. В [148] описан согласованный планировщик, основанный на эвристическом поиске в пространстве доверительных состояний, который представляет собой результат дальнейшего развития идей, впервые реализованных в 1960-х годах в частично наблюдаемых марковских процессах принятия решения, или POMDP (Partially Observable Markov Decision Processes) (см. главу 17). В настоящее время в самых быстрых согласованных планировщиках, осуществляющих поиск в пространстве доверительных состояний, таких как HSCP [118], для представления доверительных состояний используются двоичные диаграммы решений (Binary Decision Diagram — BDD) [200]; такие планировщики характеризуются быстродействием, примерно на пять порядков превышающим быстродействие алгоритма CGP.

Планировщик Warplan-C [1555] представляет собой один из вариантов планировщика Warplan, который принадлежит к числу первых планировщиков, основанных на использовании условных действий (*conditional action*). В [1153] описаны основные проблемы, касающиеся условного планирования.

Подход к организации условного планирования, описанный в данной главе, основан на эффективных алгоритмах поиска для циклических графов AND-OR, разработанных Хименесом и Торрасом [737], а также Хансеном и Зильберштейном [613]. В [119] описан подход на основе BDD, позволяющий составлять условные планы с циклами. В программе C-Buridan [413] осуществляется условное планирование для действий с вероятностными результатами; это — та же проблема, попытки решения которой предпринимались с помощью процессов POMDP (см. главу 17).

Существует тесная связь между условным планированием и автоматизированным синтезом программ; большое количество ссылок по этой теме приведено в главе 9. Но работы в этих двух научных областях осуществлялись отдельно из-за невероятной разницы в стоимостях между выполнением машинных команд и осуществлением действий с помощью транспортных средств или манипуляторов роботов. В [934] предпринята явная попытка добиться взаимного обогащения идеями между этими двумя областями.

Теперь в ретроспективе можно непредвзято наблюдать за тем, как появление основных классических алгоритмов планирования привело к созданию расширенных версий этих алгоритмов для проблемных областей, связанных с учетом неопределенности. Разработка методов на основе поиска привела к созданию методов поиска в пространстве доверительных состояний [148]; опыт в области использования алгоритма SATplan привел к созданию стохастического алгоритма SATplan [970] и к разработке средств планирования с применением булевых логических выражений с кванторами [1289]; работы в области планирования с частичным упорядочением привели к созданию программ UWL [446], CNLP [1206] и Cassandra [1241]. Опыт эксплуатации алгоритма Graphplan привел к появлению программы Sensory Graphplan, или SGP [1569], но задача разработки полностью вероятностного алгоритма Graphplan еще не совсем решена.

Одной из самых первых значительных работ по контролю выполнения было создание программы Planex [465], которая применялась в сочетании с планировщиком Strips для управления роботом Shakey. В программе Planex использовались треугольные таблицы (по сути — это эффективный механизм хранения для предусловий плана в каждом пункте плана), что обеспечивает возобновление работы после частичного неудачного завершения при выполнении плана без полного перепланирования. Модель выполнения, применяемая в роботе Shakey, рассматривается более подробно в главе 25. В планировщике Nasl [1021] проблема планирования рассматривается просто как спецификация выполнения сложного действия, что позволяет полностью унифицировать этапы выполнения и планирования. В этом планировщике для формирования рассуждений о таких сложных действиях используются средства доказательства теорем.

Одним из первых планировщиков, в котором осуществлен систематический подход к решению задач перепланирования, была программа Sipe (System for Interactive Planning and Execution monitoring) [1593], [1594]. Эта программа использовалась в демонстрационных проектах в нескольких проблемных областях, включая планирование операций на взлетной палубе авианосца и составление производственного расписания для одного из пивзаводов в Австралии. Еще в одном исследовании программа Sipe применялась для планирования строительства многоэтажных зданий, что представляет собой одну из наиболее сложных проблемных областей, в которых когда-либо применялись программы-планировщики.

Система Ipm (Integrated Planning, Execution, and Monitoring) [25] оказалась первой системой, в которой было применено сочетание планирования с частичным упорядочением и выполнения для создания непрерывно планирующего агента. В алгоритме Continuous-POP-Agent, приведенном в этой книге, объединены идеи, взятые из системы Ipm, планировщика Puccini [570] и системы Cypress [1595].

В середине 1980-х годов некоторые специалисты считали, что планирование с частичным упорядочением и связанные с ним методы никогда не позволят добиться достаточно высокого быстродействия для формирования эффективного поведения агента в реальном мире [7]. Вместо этого были предложены системы **реактивного планирования**; в своей основной форме эти системы представляли собой рефлексных агентов, возможно, с учетом внутреннего состояния, которые могут быть реализованы с помощью любого из самых различных представлений для правил “условие–действие”. В архитектуре обобщения Брукса [189] (см. главы 7 и 25) использовались многоуровневые конечные автоматы для управления движениями и

обхода препятствий шагающими и колесными роботами. Система Pengi [7] обладала способностью играть в (полностью наблюдаемую) видеоигру с использованием логических схем в сочетании с “визуальным” представлением текущих целей и внутреннего состояния агента.

В качестве метода поиска по таблице для реактивного планирования были разработаны “универсальные планы” Шопперса [1366], но, как оказалось, они представляли собой повторное открытие идеи ~~правил~~, которые уже давно использовались в марковских процессах принятия решений. Универсальный план (или правило) содержит отображение из любого состояния в действие, которое должно быть выполнено в этом состоянии. Гинсберг [557] предпринял яростную эмоциональную атаку на идею универсальных планов и, в частности, показал неразрешимость результатов для некоторых формулировок задачи реактивного планирования. Шопперс [1367] ответил на эту статью не менее эмоционально.

Как часто случается, сложившееся противоречие удалось разрешить с помощью гибридного подхода. Использование тщательно разработанных иерархий, планировщиков HTN, таких как PRS [542] и RAP [471], а также непрерывно планирующих агентов позволяет добиться сокращения времени отклика, свойственного реактивным подходам, и осуществления сложного долгосрочного планирующего поведения во многих проблемных областях.

Популярность мультиагентного планирования особенно возросла в последние годы, но само это направление имеет долгую историю. В [833] предложена формализация мультиагентного планирования в логике первого порядка, а описание в стиле Strips дано в [1195]. Понятие совместного намерения, которое приобретает особую важность, если агенты должны выполнить совместный план, впервые было сформулировано в исследованиях по актам общения [276], [277]. Приведенное в данной главе описание многотельного планирования с частичным упорядочением основано на [157].

В этой главе была лишь слегка затронута тема согласования в мультиагентном планировании. В [424] обсуждается вопрос о том, как может осуществляться совместное использование задач агентами с помощью согласования действий между ними. В [858] описана система для ведения Diplomacy, настольной игры, в которой допускается согласование действий, формирование и разрушение коалиций, а также нарушение взятых на себя обязательств. В [1468] показано, что агенты могут взаимодействовать как члены одной команды в конкурентной, динамической, частично наблюдаемой среде робототехнического футбола. В [1564] приведен обзор мультиагентных систем, который занимает целую книгу.

Модель орнитоида, приведенная на с. 609, разработана Рейнольдсом [1284], который получил премию Оскар Американской академии киноискусства за успешное применение этой модели во время съемок полчищ летучих мышей и стай пингвинов в кинофильме “Batman Returns” (Бэтмен возвращается).

## УПРАЖНЕНИЯ

**12.1.** Тщательно изучите представление времени и ресурсов, приведенное в разделе 12.1.

- Почему целесообразно сделать предикат *Duration(d)* одним из результатов действия, а не предусмотреть в действии отдельное поле в форме

*Duration: d?* (*Подсказка.* Рассмотрите условные результаты и дизъюнктивные результаты.)

- 6) Почему *Resource: m* задано как отдельное поле в действии, а не оформлено как результат?
- 12.2. ~~❖~~ **Потребляемым ресурсом** называется ресурс, который (частично) расходуется при выполнении действия. Например, для крепления двигателей к автомобилям требуются винты. Винты после их использования становятся недоступными для создания других креплений.
- Объясните, как модифицировать представление, приведенное в листинге 12.2, чтобы в нем первоначально было 100 винтов, для двигателя  $E_1$  требовалось 40 винтов, а для двигателя  $E_2$  — 50. В литералах результата для ресурсов могут использоваться функциональные символы + и -.
  - Объясните, как должно быть модифицировано определение **конфликта** между причинными связями и действиями в планировании с частичным упорядочением для учета потребляемых ресурсов.
  - Некоторые действия (например, пополнение на заводе запасов винтов или повторная заправка топливом автомобиля) способны увеличивать доступность ресурсов. Если ни одно действие не увеличивает доступность ресурса, он становится монотонно невозрастающим. Объясните, как можно использовать это свойство для отсечения ветвей в пространстве поиска.
- 12.3. Приведите декомпозиции для этапов *HireBuilder* и *GetPermit*, показанных на рис. 12.4, и объясните, как декомпонованные субпланы соединяются в общий план.
- 12.4. Приведите пример двух абстрактных субпланов из проблемной области строительства дома, которые не могут быть объединены в согласованный план без совместно выполняемых этапов. (*Подсказка.* Участками, в которых обычно должны взаимодействовать два субплана, являются те места, в которых совместно устанавливаются две физические детали дома.)
- 12.5. Некоторые специалисты утверждают, что одним из преимуществ планирования HTN является то, что оно позволяет решать задачи типа “совершить прямую и обратную поездку из Лос-Анджелеса в Нью-Йорк”, которые трудно выразить в системах обозначения, отличных от HTN, поскольку начальное и целевое состояния должны быть одинаковыми (*At(LA)*). Можете ли вы предложить способ отображения и решения таких задач без сетей HTN?
- 12.6. Покажите, как может быть перезаписано стандартное описание действия *Strips* в виде декомпозиции HTN с использованием обозначения *Achieve(p)* для указания на деятельность по достижении условия *p*.
- 12.7. Некоторые операции в стандартных языках программирования можно промоделировать как действия, которые меняют состояние мира. Например, в операции присваивания копируется содержимое участка памяти, а в операции печати изменяется состояние выходного потока. Программа, состоящая из этих операций, может также рассматриваться как план, цель которого задана в спецификации программы. Поэтому алгоритмы планирования могут использоваться для создания программ, которые реализуют заданную спецификацию.

- a) Запишите операторную схему для оператора присваивания (в котором значение одной переменной присваивается другой). Помните, что первоначальное значение должно быть при этом перезаписано!
- б) Покажите, как может использоваться в планировщике процедура создания объекта для разработки плана обмена значениями между двумя переменными с применением временной переменной.
- 12.8.** Ознакомьтесь со следующими доводами: “В инфраструктуре, которая допускает наличие неопределенных начальных состояний, **дизъюнктивные результаты** являются просто удобным способом обозначения, а не источником дополнительной выразительной мощи. Для любой схемы действий  $a$  с дизъюнктивным результатом  $P \vee Q$  можно всегда заменить его условным результатом **when**  $R$ :  $P \wedge$  **when**  $\neg R$ :  $Q$ , который, в свою очередь, можно свести к двум обычным действиям. Высказывание  $R$  обозначает случайное высказывание, которое не известно в начальном состоянии и для которого не предусмотрено каких-либо действий по получению информации с помощью датчиков”. Являются ли эти доводы правильными? Рассмотрите отдельно два случая, в одном из которых в плане имеется только один экземпляр схемы действия  $a$ , тогда как во втором имеется больше одного экземпляра.
- 12.9.** Почему условное планирование не позволяет справиться с неограниченной недетерминированностью?
- 12.10.** В мире блоков мы были вынуждены ввести два действия *Strips* (*Move* и *MoveToTable*), чтобы иметь возможность сопровождать предикат *Clear* должным образом. Покажите, как можно использовать условные результаты для представления обоих этих случаев с помощью одного действия.
- 12.11.** Условные результаты были проиллюстрированы на примере действия *Suck* в мире пылесоса — от того, в каком квадрате находится робот, зависит, какой из квадратов станет чистым. Можете ли вы предложить новое множество пропозициональных переменных для определения состояний мира пылесоса, таких, что действие *Suck* имеет безусловное описание? С использованием этих пропозициональных переменных составьте описания действий *Suck*, *Left* и *Right* и покажите, что их достаточно для представления всех возможных состояний мира.
- 12.12.** Составьте полное описание действия *Suck* для пылесоса в мире “двойного замка Мэрфи”, который иногда оставляет мусор после его перемещения в чистый квадрат назначения, а иногда оставляет мусор после выполнения действия *Suck* в чистом квадрате.
- 12.13.** Найдите достаточно грязный ковер, свободный от препятствий, и очистите его пылесосом. Нарисуйте путь, проделанный чистящей головкой пылесоса, настолько точно, насколько сможете. Объясните характер этого пути, ссылаясь на формы планирования, описанные в этой главе.
- 12.14.** Приведенные ниже цитаты взяты из инструкций на задних стенках бутылок с шампунем. Обозначьте каждую из этих инструкций как безусловный план, условный план или план с контролем выполнения: а) “Разотрите пену. Прополосните волосы. Повторите процедуру”; б) “Нанесите шампунь на волосы

и оставьте на несколько минут. Прополосните волосы и повторите процедуру в случае необходимости”; в) “Если не удается устраниТЬ проблемы, обратитесь к врачу”.

- 12.15.** В алгоритме *And-Or-Graph-Search*, приведенном в листинге 12.4, осуществляется проверка повторяющихся состояний только в пути от корня до текущего состояния. Предположим, что, кроме того, в этом алгоритме было бы предусмотрено хранение в виде списка каждого когда-либо посещенного состояния и проверка по этому списку (см., например, алгоритм *Graph-Search*, приведенный в листинге 3.6). Определите, какая информация должна записываться в список и как она должна использоваться в алгоритме при обнаружении повторяющегося состояния. (*Подсказка*. Вам потребуется, по меньшей мере, проводить различие между состояниями, для которых перед этим был успешно сконструирован субплан, и состояниями, для которых не удалось найти субплана.) Объясните, как можно использовать **метки** для предотвращения необходимости иметь несколько копий субпланов.
- 12.16.** Объясните точно, как следует модифицировать алгоритм *And-Or-Graph-Search* для выработки циклического плана, если не существует ни одного ациклического плана. Вам придется рассмотреть три проблемы: применение меток для этапов плана так, чтобы циклический план мог указывать обратно на какую-то предыдущую часть плана, модификация алгоритма *Or-Search* так, чтобы он продолжал поиск ациклических планов после нахождения циклического плана, а также дополнение средств представления плана, чтобы они позволяли указать, является ли план циклическим. Покажите, как действует ваш алгоритм: а) в мире пылесоса с тройным законом Мэрфи и б) в альтернативном мире пылесоса с двойным законом Мэрфи. Вам может потребоваться реализация алгоритма на компьютере для проверки полученных результатов. Может ли план для случая б) быть записан с использованием стандартного синтаксиса цикла?
- 12.17.** Полностью определите процедуру обновления доверительного состояния для частично наблюдаемых вариантов среды. Под этим подразумевается метод вычисления нового представления доверительного состояния (как списка высказываний с оценкой знаний) из текущего представления доверительного состояния и описания действия с условными результатами.
- 12.18.** Составьте описания действий, аналогичные уравнению 12.2, для действий *Right* и *Suck*. Запишите также описание для действия *CheckLocation*, аналогичное приведенному уравнению 12.3. Повторите это упражнение, используя альтернативное множество высказываний из упр. 12.11.
- 12.19.** Рассмотрите приведенный на с. 599 список действий, которые не могут быть выполнены перепланирующим агентом. Составьте набросок алгоритма, который позволяет выполнить одно или несколько таких действий.
- 12.20.** Рассмотрите следующую задачу: пациент поступил в приемную врача с симптомами, которые могут быть вызваны либо обезвоживанием, либо болезнью *D* (но не обеими этими причинами вместе). Существуют два возможных действия: *Drink* (Питьё), которое, безусловно, позволяет справиться с обезвоживанием, и *Medicate* (Медикаментозное лечение), излечивающее заболевание

*D*, но имеющее нежелательный побочный эффект, который возникает, если организм пациента обезвожен. Составьте описание этой проблемы на языке PDDL и разработайте план без использования датчиков, позволяющий решить эту задачу, перечислив все относящиеся к ней возможные миры.

- 12.21.** В задаче из области медицины, приведенной в предыдущем упражнении, добавьте действие *Test* (Лабораторный анализ), которое имеет условный результат *CultureGrowth* (Развитие бактериальной культуры), если предикат *Disease* имеет истинное значение, и в любом случае имеет результат из области восприятия *Known(CultureGrowth)*. Составьте условный план, который решает эту задачу и сводит к минимуму использование действия *Medicate*.

## Часть V

# НЕОПРЕДЕЛЕННЫЕ ЗНАНИЯ И РАССУЖДЕНИЯ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ

Неопределенность	622
Вероятностные рассуждения	660
Вероятностные рассуждения во времени	718
Принятие простых решений	778
Принятие сложных решений	815

# 13 НЕОПРЕДЕЛЕННОСТЬ

*В данной главе показано, как должен действовать агент в условиях недостающей информации.*

## 13.1. ДЕЙСТВИЯ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ

Логические агенты, описанные в частях III и IV, пользовались базами знаний, эпистемологический вклад которых состоял в том, что содержащиеся в них высказывания были истинными, ложными или неизвестными. Если агенту известно достаточно фактов о его среде, соответствующий логический подход позволяет ему формировать планы, которые гарантированно будут работать. Такая организация функционирования агента является очень удобной. К сожалению, ~~агенты почти никогда не имеют доступа ко всем необходимым сведениям о своей среде~~. Поэтому агенты должны действовать в условиях **неопределенности**. Например, агент в мире вампуза, описанный в главе 7, имеет датчики, которые сообщают ему только локальную информацию; основная часть мира не является для него непосредственно наблюдаемой. Агент в мире вампуза часто оказывается в такой ситуации, что не имеет возможности определить, какой из двух квадратов содержит яму. Если эти квадраты находятся на пути к золоту, то агенту может потребоваться испытать судьбу и войти наугад в один из этих двух квадратов.

Реальный мир намного сложнее по сравнению с миром вампуза. Логический агент не всегда имеет возможность составить полное и правильное описание того, как будут осуществляться его действия. Предположим, например, что агент-водитель такси желает отвести пассажира в аэропорт, чтобы тот успел на самолет, и составляет план  $A_{90}$ , в котором предусматривается выезд из дома за 90 минут до вылета самолета и вождение на приемлемой скорости. Но даже если аэропорт находится от дома на расстоянии примерно 15 миль, агент все равно не сможет со всей определенностью сделать вывод, что “план  $A_{90}$  позволит нам прибыть в аэропорт вовремя”. Вместо этого он приходит к более слабому заключению: “План  $A_{90}$  позволит нам прибыть в аэропорт вовремя, если только в моем автомобиле не возникнет неисправность или не закончится топливо, и я не попаду в аварию, и не будет аварий на мосту, и самолет не вылетит раньше времени, и...”. Ни одно из этих условий нельзя довести до логического вывода, поэтому невозможно сформировать логическим путем заключение о том, что реализация плана будет успешной. Это — один из примеров **проблемы спецификации**, которая упоминалась в главе 10.

Если логический агент не сможет сделать заключение о том, какая именно конкретная стратегия позволяет ему достичь цели, то не сможет действовать. Условное планирование позволяет до некоторой степени преодолеть неопределенность, но только если агент сможет получать требуемую информацию с помощью действий, предусматривающих применение датчиков, и только если количество различных непредвиденных ситуаций не слишком велико. Еще одно возможное решение состоит в том, чтобы наделить агента простой, но, возможно, неправильной теорией мира, которая позволит ему составить план; можно надеяться на то, что подобные планы чаще всего будут выполнимыми, но если происходящие события будут противоречить теории агента, возникнут проблемы. Более того, даже сам выбор компромисса между точностью и полезностью этой рабочей теории агента, по-видимому, потребует формирования рассуждений о неопределенности. В конечном итоге ни один чисто логический агент не сможет прийти к выводу, что план  $A_{90}$  — именно то, что следует делать.

Тем не менее предположим, что план  $A_{90}$  действительно представляет собой правильное руководство к действию. Что подразумевается под этим утверждением? Как было описано в главе 2, под этим подразумевается, что из всех планов, которые могут быть выполнены, только план  $A_{90}$ , по всей видимости, позволит максимизировать показатели производительности агента при наличии всей той информации о своей среде, которую имеет агент. Показатели производительности включают своевременную доставку пассажира в аэропорт к указанному рейсу, предотвращение продолжительного, непродуктивного ожидания в аэропорту и избавление от необходимости дополнительно оплачивать проезд по скоростным участкам шоссе. Информация, которой обладает агент, не позволяет гарантировать получение ни одного из этих результатов выполнения плана  $A_{90}$ , но может обеспечить достижение некоторой степени уверенности в том, что эти результаты будут достигнуты. Другие планы, такие как  $A_{120}$ , способны повысить степень уверенности агента в том, что он доставит пассажира в аэропорт вовремя, но вместе с тем повысят вероятность продолжительного ожидания.  $\bowtie$  Таким образом, выбор правильной стратегии — рационального решения — зависит и от относительной важности различных целей, и от вероятности того (от степени уверенности в том), что они могут быть достигнуты. Эти идеи будут уточнены в оставшейся части данного раздела в целях подготовки к разработке общих теорий проведения рассуждений в условиях неопределенности и принятия рациональных решений, которые будут представлены в этой и последующих главах.

### Учет наличия неопределенных знаний

В этом разделе характер неопределенных знаний рассматривается более подробно. Для иллюстрации связанных с этим понятий будет использоваться простой пример из области диагностики. Диагностика (проводимая при обследовании пациента, при ремонте автомобиля или в других областях) представляет собой задачу, в которой почти всегда приходится сталкиваться с неопределенностью. Попробуем записать правила для диагностики заболеваний зубов с использованием логики первого порядка, чтобы можно было ознакомиться с тем, какие трудности возникают при осуществлении логического подхода. Рассмотрим следующее правило:

$$\forall p \ Symptom(p, \text{Toothache}) \Rightarrow Disease(p, \text{Cavity})$$

Проблема состоит в том, что это правило является неверным. Не все пациенты с зубной болью имеют в зубе дупло; у некоторых из них встречается заболевание десен, нарыв или одна из нескольких других проблем:

$$\forall p \ Symptom(p, Toothache) \Rightarrow \\ Disease(p, Cavity) \vee Disease(p, GumDisease) \vee Disease(p, Abscess) \dots$$

К сожалению, для того чтобы сделать это правило истинным, мы вынуждены ввести в него почти бесконечный список возможных причин. Еще одна попытка может состоять в том, чтобы это правило было преобразовано в причинное правило:

$$\forall p \ Disease(p, Cavity) \Rightarrow Symptom(p, Toothache)$$

Но и это правило нельзя назвать верным; не все зубы, в которых имеется дупло, вызывают боль. Единственный способ откорректировать данное правило состоит в том, чтобы сделать его логически исчерпывающим: дополнить левую сторону описаниями всех обстоятельств, которые должны иметь место для того, чтобы дупло вызывало зубную боль. Но даже в этом случае с целью правильной диагностики необходимо также учитывать вероятность того, что наличие зубной боли у пациента и наличие дупла в зубе могут быть не связаны.

Таким образом, попытка использовать логику первого порядка для представления знаний в таких проблемных областях, как медицинская диагностика, оканчивается неудачей по трем основным причинам, описанным ниже.

- **✗ Экономия усилий.** Для формирования полного множества антецедентов или консеквентов, необходимого для составления правила, не имеющего исключений, требуется слишком много работы, а само применение таких правил является слишком сложным.
- **✗ Отсутствие теоретических знаний.** Медицинская наука не имеет полной теории для данной проблемной области.
- **✗ Отсутствие практических знаний.** Даже если известны все правила, может оставаться неопределенность в отношении диагноза данного конкретного пациента, поскольку все необходимые обследования не были или не могли быть выполнены.

Связь между зубной болью и наличием дупла не является просто логическим следствием, действующим в обоих направлениях. Такая ситуация типична не только для данной области медицинской диагностики, но и для большинства других областей, в которых требуется формирование суждений об определенных ситуациях: юриспруденция, экономика, проектирование, ремонт автомобилей, садоводство, датирование ископаемых находок и т.д. Знания агента в лучшем случае позволяют сформировать относящиеся к делу высказывания только с определенной **✗ степенью уверенности** (degree of belief). Основным применяемым нами инструментальным средством для учета степеней уверенности будет **✗ теория вероятностей**, в которой каждому высказыванию присваивается числовое значение степени уверенности от 0 до 1. (Некоторые альтернативные методы формирования рассуждений в условиях неопределенности описаны в разделе 14.7.)

**✗ Вероятности предоставляют способ суммарного учета неопределенности, возникающей по причинам экономии усилий и отсутствия знаний.** Мы не можем знать со всей уверенностью, что беспокоит данного конкретного пациента, но можем быть уверенны-

ми в том, что, скажем, в 80 случаях из 100 (т.е. с вероятностью 0,8) у пациента в зубе имеется дупло, если он испытывает зубную боль. Это означает, что из числа всех ситуаций, неотличимых от текущей ситуации в рамках тех знаний, которыми обладает агент, пациент в 80% этих ситуаций должен иметь дупло в зубе. Такая уверенность может быть основана на статистических данных (о том, что у 80% пациентов с зубной болью, наблюдавшихся до сих пор, было обнаружено дупло в зубе), или на основе некоторых общих правил, или с использованием определенной комбинации сведений, полученных из разных источников. В этих 80% дана сводная информация обо всех случаях, в которых присутствовали все факторы, необходимые для того, чтобы дупло вызывало зубную боль, и о других случаях, в которых у пациента были и дупло, и зубная боль, но эти два обстоятельства оказались несвязанными. Эти недостающие 20% подытоживают все другие возможные причины зубной боли, для подтверждения или отрицания которых мы либо затратили слишком мало усилий, либо не имели достаточно знаний.

Присваивание вероятности 0 данному конкретному высказыванию соответствует безусловной уверенности в том, что это высказывание ложно, а присваивание вероятности 1 соответствует безусловной уверенности, что высказывание истинно. Значения вероятности между 0 и 1 соответствуют промежуточным степеням уверенности в истинности высказывания. В действительности само высказывание может быть либо истинным, либо ложным независимо от этого. Важно отметить, что степень уверенности отличается от степени истинности. Вероятность 0,8 не означает “истинно на 80%”, а просто указывает на 80%-ную степень уверенности, т.е. на довольно обоснованные ожидания. Таким образом, теория вероятностей вносит такой же онтологический вклад, как и логика, т.е. позволяет указать, являются ли некоторые факты действительными в этом мире. Степени истинности, в отличие от степеней уверенности, являются предметом **нечеткой логики**, которая рассматривается в разделе 14.7.

В логике такое высказывание, как “У пациента в зубе имеется дупло”, является истинным или ложным в зависимости от интерпретации и от мира; оно истинно именно тогда, когда имеет место факт, на который оно ссылается. С другой стороны, в теории вероятностей такое высказывание, как “Вероятность того, что у данного пациента в зубе имеется дупло, равна 0,8”, касается степени уверенности агента, а не относится непосредственно к самому миру. Эта степень уверенности зависит от результатов восприятия, полученных агентом до сих пор. Сами результаты восприятия представляют собой **свидетельство**, на котором основаны вероятностные утверждения. Например, предположим, что агент вытянул карту из растасованной колоды. Прежде чем посмотреть на эту карту, агент может присвоить значение вероятности 1/52 такому событию, что карта окажется тузом пик, а после взгляда на вынутую из колоды карту соответствующая вероятность для того же высказывания примет значение 0 или 1. Таким образом, присваивание значения вероятности некоторому высказыванию аналогично утверждению о том, что данное конкретное логическое высказывание (или его отрицание) следует из базы знаний, а не о том, является ли оно истинным или ложным. Разумеется, оценка того, следует ли высказывание из базы знаний, может изменяться по мере добавления в базу знаний новых высказываний; и, по аналогии с этим, вероятности могут изменяться после получения дополнительных свидетельств<sup>1</sup>.

<sup>1</sup> Эта ситуация полностью отличается от той, в которой некоторое высказывание становится истинным или ложным по мере того, как изменяется мир. Для учета изменений в мире с помощью вероятностей требуются механизмы такого же рода (ситуации, интервалы и события), которые использовались в главе 10 для логических представлений. Эти механизмы рассматриваются в главе 15.

Поэтому во всех вероятностных утверждениях должно быть указано свидетельство, с учетом которого оценивалась данная вероятность. По мере получения агентом новых результатов восприятия его вероятностные оценки обновляются таким образом, чтобы в них отражались новые свидетельства. Вероятности, оцениваемые до получения свидетельства, называются **априорными**, или **безусловными**, вероятностями, а вероятности, оцениваемые после получения свидетельства, называются **апостериорными**, или **условными**, вероятностями. В большинстве случаев агент должен получать определенные свидетельства из результатов своих восприятий; после этого ему необходимо вычислять апостериорные вероятности результатов, которые его интересуют.

### Неопределенность и рациональные решения

Из-за наличия неопределенности способ принятия решений агентом изменяется коренным образом. В обычных условиях логический агент ставит перед собой цель и выполняет любой план, который гарантирует ее достижение. Действие в этом плане может быть выбрано или отвергнуто с учетом того, способствует ли оно достижению цели, независимо от наличия или отсутствия каких-либо иных действий, способствующих ее достижению. А если в ситуацию вмешивается неопределенность, такой подход становится неосуществимым. Снова рассмотрим план прибытия в аэропорт,  $A_{90}$ . Предположим, что этот план имеет 95%-ные шансы на успех. Означает ли это, что решение по выбору данного плана является рациональным? Не обязательно: могут существовать другие планы, такие как  $A_{120}$ , с большими вероятностями успеха. Если для пассажира жизненно важно успеть на самолет, то стоит рискнуть тем, что ему придется дольше ждать в аэропорту. А что можно сказать о плане  $A_{1440}$ , который предусматривает заблаговременный выезд из дома за 24 часа до отправления самолета? В большинстве обстоятельств это — не лучший выбор, поскольку он предусматривает невыносимо долгое ожидание, даже несмотря на то, что почти полностью гарантирует своевременное прибытие в аэропорт.

Чтобы иметь возможность выбирать среди подобных вариантов, агент должен вначале получить информацию о **предпочтениях** между различными возможными **результатами** разных планов. Каждый конкретный результат представляет собой полностью определенное состояние, включая такие факторы, как своевременное прибытие агента и продолжительность ожидания в аэропорту. Для представления и формирования рассуждений с учетом предпочтений мы будем использовать **теорию полезности**. (Термин “полезность” имеет англоязычный эквивалент “utility”, который в данном контексте обозначает “свойство быть полезным”, а не электростанцию или предприятие, предоставляющее коммунальные услуги.) Теория полезности указывает, что каждое состояние имеет определенную степень полезности (или просто **полезность**) для агента и что агент предпочитает состояние с более высокой полезностью.

Полезность состояния является относительной для агента, предпочтения которого должна описывать функция полезности. Например, функции вознаграждения для игр, описанные в главе 6, представляют собой функции полезности. Полезность состояния, в котором белые могут победить в ходе какой-то шахматной партии, безусловно, высока для агента, играющего белыми, но низка для агента, играющего черными. Еще один пример состоит в том, что некоторые игроки (включая авторов

этой книги) будут счастливы, сыграв вничью против чемпиона мира, а о других играх (включая бывшего чемпиона мира) этого сказать нельзя. При этом не учитываются личные вкусы или предпочтения: читатель может подумать, что агент, который предпочитает шоколадным чипсам мороженое “Халапеньо” с добавлением компонентов жевательной резинки, — странный или даже бестолковый тип, но не сможет утверждать, что этот агент нерационален. В функции полезности может быть даже учтена польза от альтруистического поведения просто путем включения оценки благополучия других как одного из факторов, которые вносят вклад в полезность для самого агента.

Предпочтения, будучи выражеными в виде полезности, комбинируются с вероятностями в общей теории рациональных решений, называемой **теорией решений**, следующим образом:

$$\text{Теория решений} = \text{Теория вероятностей} + \text{Теория полезности}$$

Фундаментальная идея теории решений состоит в том, что ~~любой агент является рациональным тогда и только тогда, когда он выбирает действие, позволяющее достичь наибольшей ожидаемой полезности, усредненной по всем возможным результатам данного действия~~. Это — так называемый принцип **максимальной ожидаемой полезности** (Maximum Expected Utility — MEU). Мы наблюдали этот принцип в действии в главе 6, когда кратко рассматривали оптимальные решения в наардах, а ниже будет показано, что это — действительно полностью общий принцип.

### Проект агента, действующего в соответствии с теорией решений

Набросок структуры агента, который использует теорию решений для выбора действий, приведен в листинге 13.1. На некотором уровне абстракции этот агент идентичен логическому агенту, описанному в главе 7. Основное различие состоит в том, что знания о текущем состоянии агента, действующего в соответствии с теорией решений, являются неопределенными; ~~доверительное состояние~~ этого агента является представлением вероятностей всех возможных фактических состояний мира. Со временем агент накапливает больше свидетельств и его доверительное состояние изменяется. На основании своего доверительного состояния агент может делать вероятностные предсказания результатов действий и поэтому выбирать действие с наивысшей ожидаемой полезностью. В настоящей и следующей главах изложение в основном сосредоточивается на задаче представления и вычисления с учетом вероятностной информации в целом. Глава 15 посвящена описанию методов решения конкретных задач представления и обновления доверительного состояния, а также задач предсказания обстановки в среде. В главе 16 более подробно рассматривается теория полезности, а в главе 17 разрабатываются алгоритмы, применяемые для принятия сложных решений.

**Листинг 13.1.** Агент, действующий на основании теории решений, который выбирает рациональные действия. Этапы работы этого алгоритма подробно описываются в следующих пяти главах

---

```
function DT-Agent(percept) returns действие action
    static: belief_state, доверительное состояние - вероятностные
            убеждения, касающиеся текущего состояния мира
    action, действие агента
```

---

```

обновить доверительное состояние belief_state с учетом действия
action и восприятия percept
вычислить результирующие вероятности для действий actions
на основании описаний действий action и текущего доверительного
состояния belief_state
выбрать действие action с наивысшей ожидаемой полезностью
с учетом вероятностей результатов и информации о полезности
return action

```

---

## 13.2. ОСНОВНАЯ ВЕРОЯТНОСТНАЯ СИСТЕМА ОБОЗНАЧЕНИЙ

Теперь, после определения общей инфраструктуры для рационального агента, нам потребуется формальный язык для представления и формирования рассуждений с неопределенными знаниями. Любая система обозначений, применяемая для описания степени уверенности, должна предоставлять возможность решать две основные проблемы: отражать характер высказываний, которым присваиваются оценки степени уверенности, и показывать зависимость степени уверенности от опыта агента. В представленной здесь версии теории вероятностей используется одно из расширений пропозициональной логики, которое распространяется на высказывания в этой логике. Зависимость от опыта отражается в синтаксическом различии между априорными вероятностными утверждениями, которые применяются до получения каких-либо свидетельств, и условными вероятностными утверждениями, которые явно включают соответствующие свидетельства.

### Высказывания

Оценки степеней уверенности всегда применяются к **высказываниям** — утверждениям о том, что имеет место *то-то* и *то-то*. До сих пор в этой книге рассматривались два формальных языка, применяемых для составления высказываний, — пропозициональная логика и логика первого порядка. В теории вероятностей, как правило, используется язык, немного более выразительный, чем пропозициональная логика. Этот язык описан в данном разделе. (В разделе 14.6 обсуждаются способы, позволяющие предписывать оценки степеней уверенности утверждениям в логике первого порядка.)

Основным элементом этого языка является **случайная переменная**, которая может рассматриваться как ссылающаяся на некоторую “часть” мира, “состояние” которого первоначально неизвестно. Например, утверждение *Cavity* может касаться того, имеется ли у пациента дупло в нижнем левом зубе мудрости. Случайные переменные играют роль, аналогичную той, которую выполняют переменные CSP в задачах удовлетворения ограничений и пропозициональные символы в пропозициональной логике. Мы будем всегда записывать имена случайных переменных с прописной буквы. (Тем не менее для представления любой неизвестной случайной переменной все еще используются однобуквенные имена в виде строчных букв, например  $P(a) = 1 - P(\neg a)$ .)

Каждая случайная переменная имеет **область определения** значений, которые она может принимать. Например, область определения переменной *Cavity* может

представлять собой<sup>2</sup>  $\langle \text{true}, \text{false} \rangle$ . (Для имен значений используются прописные буквы.) В этом высказывании простейшего вида утверждается, что случайная переменная имеет конкретное значение, взятое из ее области определения. Например,  $\text{Cavity} = \text{true}$  может представлять высказывание, что у пациента действительно имеется дупло в левом нижнем зубе мудрости.

Как и в случае переменных CSP, случайные переменные обычно подразделяются на три описанных ниже вида, в зависимости от типа области определения.

- **Булевые случайные переменные**, такие как  $\text{Cavity}$ , имеют область определения  $\langle \text{true}, \text{false} \rangle$ . Мы часто будем сокращенно записывать высказывание, подобное  $\text{Cavity}=\text{true}$ , в виде имени этой переменной, записанного со строчной буквой,  $\text{cavity}$ . Аналогичным образом, высказывание  $\text{Cavity}=\text{false}$  будет сокращенно записываться как  $\neg\text{cavity}$ .
- **Дискретные случайные переменные**, которые включают булевые случайные переменные как частный случай и принимают значения из счетной области определения. Например, областью определения погоды  $\text{Weather}$  может быть  $\langle \text{sunny}, \text{rainy}, \text{cloudy}, \text{snow} \rangle$ . Значения в области определения должны быть взаимно исключительными и исчерпывающими. Если не возникает путаница, мы будем, например, использовать  $\text{snow}$  как сокращение для высказывания  $\text{Weather}=\text{snow}$ .
- **Непрерывные случайные переменные** принимают свои значения из области действительных чисел. Эта область определения может представлять собой всю ось действительных чисел или некоторое ее подмножество, такое как интервал  $[0, 1]$ . Например, в высказывании  $X=4.02$  утверждается, что случайная переменная  $X$  имеет точное значение 4.02. Высказывания, касающиеся непрерывных случайных переменных, могут также представлять собой неравенства, такие как  $X \leq 4.02$ .

За небольшими исключениями, мы будем в основном сосредоточиваться на дискретном случае.

Элементарные высказывания, такие как  $\text{Cavity}=\text{true}$  и  $\text{Toothache}=\text{false}$ , могут комбинироваться для формирования сложных высказываний с использованием всех стандартных логических связок. Например,  $\text{Cavity} = \text{true} \wedge \text{Toothache} = \text{false}$  представляет собой высказывание, которому может быть предписана определенная степень уверенности (неуверенности). Как было указано в предыдущем абзаце, это высказывание может быть также записано как  $\text{cavity} \wedge \neg\text{toothache}$ .

## Атомарные события

Для понимания основ теории вероятностей полезно ознакомиться с понятием **атомарного события**. Атомарное событие представляет собой полную спецификацию состояния мира, в отношении которого знания агента являются неопределенными. Оно может рассматриваться как некоторое присваивание конкретных значе-

<sup>2</sup> Читатель мог бы предположить, что эту область определения следует записывать как множество:  $\{\text{true}, \text{false}\}$ . Но авторы записывают ее в виде кортежа, поскольку в дальнейшем это позволит упростить наложение упорядоченности на значения.

ний всем переменным, из которых состоит этот мир. Например, если мир пациента состоит только из булевых переменных *Cavity* и *Toothache*, то существует всего лишь четыре разных атомарных события<sup>3</sup>; одним из таких событий является высказывание *Cavity=false*  $\wedge$  *Toothache=true*.

Атомарные события имеют некоторые важные свойства, описанные ниже.

- Они являются взаимно исключающими — фактически может иметь место, самое большое, одно такое событие. Например, не могут одновременно происходить такие события, как *cavity*  $\wedge$  *toothache* и *cavity*  $\wedge$   $\neg$ *toothache*.
- Множество всех возможных атомарных событий является исчерпывающим — должно иметь место по меньшей мере одно из этих событий. Это означает, что дизъюнкция всех атомарных событий логически эквивалентна *true*.
- Из любого конкретного атомарного события следует истинность или ложность каждого высказывания, либо простого, либо сложного. В этом можно убедиться, используя стандартные определения семантики для логических связок (см. главу 7). Например, из атомарного события *cavity*  $\wedge$   $\neg$ *toothache* следует истинность высказывания *cavity* и ложность высказывания *cavity*  $\Rightarrow$  *toothache*.
- Любое высказывание логически эквивалентно дизъюнкции всех атомарных событий, из которых следует истинность этого высказывания. Например, высказывание *cavity* эквивалентно дизъюнкции атомарных событий *cavity*  $\wedge$  *toothache* и *cavity*  $\wedge$   $\neg$ *toothache*.

В упр. 13.4 предлагается доказать некоторые из этих свойств.

## Априорная вероятность

❖ Безусловная, или ❖ априорная, вероятность, связанная с высказыванием *a*, представляет собой степень уверенности, относящуюся к этому высказыванию в отсутствии любой другой информации; она записывается как *P(a)*. Например, если априорная вероятность того, что у пациента в зубе имеется дупло, равна 0,1, то можно записать следующее:

$$P(\text{Cavity} = \text{true}) = 0.1 \text{ или } P(\text{cavity}) = 0.1$$

Важно помнить, что вероятность *P(a)* может использоваться, только если нет другой информации. Как только становится известной какая-то новая информация, мы должны проводить рассуждения с условной вероятностью высказывания *a*, в которой учитывается эта новая информация. Условные вероятности рассматриваются в следующем разделе.

Иногда приходится вести речь о вероятностях всех возможных значений случайной переменной. В этом случае используется такое выражение, как *P(Weather)*, которое обозначает вектор значений для вероятностей каждого отдельного состоя-

<sup>3</sup> Во многих стандартных определениях теории вероятностей в качестве примитивных рассматриваются атомарные события, известные также под названием **элементов выборки**, а случайная переменная задается как функция, принимающая атомарное событие в качестве параметра и возвращающая значение из соответствующей области определения. Возможно, что такой подход является более общим, но вместе с тем он менее понятен интуитивно.

ния погоды. Таким образом, вместо того, чтобы записывать следующие четыре уравнения:

$$\begin{aligned}P(\text{Weather} = \text{sunny}) &= 0.7 \\P(\text{Weather} = \text{rain}) &= 0.2 \\P(\text{Weather} = \text{cloudy}) &= 0.08 \\P(\text{Weather} = \text{snow}) &= 0.02\end{aligned}$$

можно просто применить такую запись:

$$\mathbf{P}(\text{Weather}) = \langle 0.7, 0.2, 0.08, 0.02 \rangle$$

В этом выражении определено **распределение априорных вероятностей** для случайной переменной *Weather*.

Кроме того, такие выражения, как  $\mathbf{P}(\text{Weather}, \text{Cavity})$ , могут использоваться для обозначения вероятностей всех комбинаций значений множества случайных переменных<sup>4</sup>. В этом случае выражение  $\mathbf{P}(\text{Weather}, \text{Cavity})$  можно представить с помощью таблицы вероятностей с размерами  $4 \times 2$ . Такая таблица называется **совместным распределением вероятностей** переменных *Weather* и *Cavity*.

Иногда возникает необходимость рассматривать полное множество случайных переменных, используемых для описания мира. Совместное распределение вероятностей, которое охватывает указанное полное множество, называется **полным совместным распределением вероятностей**. Например, если мир состоит только из переменных *Cavity*, *Toothache* и *Weather*, то полное совместное распределение определяется следующим выражением:

$$\mathbf{P}(\text{Cavity}, \text{Toothache}, \text{Weather})$$

Это совместное распределение может быть представлено в виде таблицы с размерами  $2 \times 2 \times 4$ , имеющей 16 элементов. Полное совместное распределение вероятностей задает вероятность каждого атомарного события и поэтому представляет собой полную спецификацию неопределенности знаний некоего лица о рассматриваемом мире. Как будет показано в разделе 13.4, с помощью полного совместного распределения можно получить ответ на любой запрос, касающийся вероятностных знаний.

Для непрерывных переменных возможность записать все распределение в виде таблицы просто исключена, поскольку количество значений бесконечно велико. Вместо этого обычно определяется вероятность, которую принимает случайная переменная при некотором значении *x*, в виде параметризованной функции от *x*. Например, допустим, что случайная переменная *X* обозначает прогноз максимальной температуры воздуха на завтра в Беркли. В таком случае следующее высказывание:

$$P(X = x) = U[18, 26](x)$$

выражает уверенность в том, что значение *X* распределено равномерно между 18 и 26 градусами Цельсия. (Некоторые полезные непрерывные распределения описаны в приложении А.) Вероятностные распределения для непрерывных переменных называются **функциями плотности вероятностей**. Функции плотности отличаются по смыслу от дискретных распределений. Например, с использованием распределения температур,

---

<sup>4</sup> Общее правило применения этого обозначения состоит в том, что распределение охватывает все значения переменных, имена которых записаны с прописной буквы. Таким образом, выражение  $\mathbf{P}(\text{Weather}, \text{cavity})$  — это четырехэлементный вектор вероятностей для конъюнкции каждого высказывания о состоянии погоды с высказыванием *Cavity=true*.

приведенного выше, можно найти, что  $P(X = 20.5) = U[18, 26](20.5) = 0.125/C$ . Это не означает, что имеется 12,5% шансов на то, что завтра максимальная температура будет точно равна 20,5 градуса; вероятность того, что это произойдет, разумеется, равна нулю. Формальный смысл приведенного выше выражения состоит в том, что вероятность пребывания значений температуры в небольшом регионе вокруг 20,5 градуса равна в этих пределах значению 0,125, деленному на ширину региона в градусах Цельсия:

$$\lim_{dx \rightarrow 0} P(20.5 \leq X \leq 20.5+dx) / dx = 0.125/C$$

Некоторые авторы используют разные символы для дискретных распределений и функций плотности, а авторы данной книги в обоих случаях используют обозначение  $P$ , поскольку путаница возникает редко и уравнения для обоих случаев обычно одинаковы. Следует учитывать, что вероятности обозначаются безразмерными числами, а функции плотности измеряются с помощью некоторой единицы, в данном примере в качестве такой единицы применяется единица измерения, обратная градусам Цельсия.

### Условная вероятность

После того как агент получает определенное свидетельство, касающееся ранее неизвестных случайных переменных, составляющих рассматриваемую проблемную область, априорные вероятности становятся больше не применимыми. Вместо этого должны использоваться **условные**, или **апостериорные**, вероятности. При этом используется обозначение<sup>5</sup>  $P(a|b)$ , где  $a$  и  $b$  — любые высказывания. Это обозначение читается как “вероятность  $a$ , при условии, что все, что нам известно, — это  $b$ ”. Например, следующее выражение:

$$P(\text{cavity} | \text{toothache}) = 0.8$$

показывает, что если наблюдается пациент, имеющий зубную боль, и еще не получена какая-либо иная информация, то вероятность наличия у этого пациента дупла в зубе составляет 0,8. Априорная вероятность, такая как  $P(\text{cavity})$ , может рассматриваться как частный случай условной вероятности,  $P(\text{cavity}|)$ , где условием вероятности является отсутствие свидетельства.

Условные вероятности могут быть определены в терминах безусловных вероятностей. Таким определяющим уравнением является следующее, которое остается истинным, если  $P(b) > 0$ :

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \quad (13.1)$$

Это уравнение может быть также записано следующим образом и в таком виде называется **правилом произведения**:

$$P(a \wedge b) = P(a|b)P(b)$$

По-видимому, правило произведения запомнить проще; оно основано на таком факте: для того чтобы  $a$  и  $b$  были истинными, необходимо, чтобы  $b$  было истинным,

---

<sup>5</sup> Оператор  $|$  имеет наименьший возможный приоритет, поэтому выражение  $P(a \wedge b | c \vee d)$  означает  $P((a \wedge b) | (c \vee d))$ .

а также необходимо, чтобы  $a$  было истинным, если дано  $b$ . Такое же утверждение можно выразить иначе:

$$P(a \wedge b) = P(b|a)P(a)$$

В некоторых случаях проще формировать рассуждения в терминах априорных вероятностей конъюнкций, но чаще всего мы в качестве своего основного инструмента для вероятностного логического вывода будем использовать условные вероятности.

Кроме того, для условных распределений может использоваться обозначение  $\mathbf{P}$ . Выражение  $\mathbf{P}(X|Y)$  задает значения выражения  $P(X=x_i|Y=y_j)$  для каждой возможной комбинации  $i, j$ . В качестве примера того, что с помощью векторного обозначения можно добиться сокращения размеров формул, рассмотрим, как применяется правило произведения к каждому случаю, когда высказывания  $a$  и  $b$ , соответственно, подтверждают конкретные значения переменных  $X$  и  $Y$ . При этом будут получены следующие уравнения:

$$P(X = x_1 \wedge Y = y_1) = P(X = x_1 | Y = y_1)P(Y = y_1)$$

$$P(X = x_1 \wedge Y = y_2) = P(X = x_1 | Y = y_2)P(Y = y_2)$$

...

Все эти уравнения можно объединить в такое единственное уравнение:

$$\mathbf{P}(X, Y) = \mathbf{P}(X|Y) \mathbf{P}(Y)$$

Следует учитывать, что это уравнение обозначает множество уравнений, которые связывают между собой соответствующие отдельные записи в таблицах распределения вероятностей, а не выражает матричное умножение этих таблиц.

Было бы соблазнительно, но неправильно рассматривать условные вероятности как логические следствия с оценкой неопределенности. Например, высказывание  $P(a|b)=0.8$  нельзя интерпретировать в том смысле, что “если  $b$  истинно, из этого следует вывод, что вероятность  $P(a)$  равна 0,8”. Такая интерпретация была бы неправильной в двух отношениях: во-первых,  $P(a)$  всегда обозначает априорную вероятность  $a$ , но не апостериорную вероятность, полученную с учетом некоторого свидетельства; во-вторых, само утверждение  $P(a|b) = 0.8$  непосредственно относится к делу, только если  $b$  — единственное доступное свидетельство. Как только становится доступной дополнительная информация  $c$ , степень уверенности в истинности  $a$  становится равной  $P(a|b \wedge c)$ , а это значение может быть почти не связанным со значением  $P(a|b)$ . Например, в высказывании  $c$  может быть непосредственно указано, является ли  $a$  истинным или ложным. Если врач обследует пациента, который жалуется на зубную боль, и обнаруживает дупло, то получает дополнительное свидетельство *cavity* и составляет логический вывод (триивиальный), что  $P(cavity|toothache \wedge cavity) = 1.0$ .

### ИСТОКИ ПОНЯТИЯ ВЕРОЯТНОСТИ

По поводу того, каковы истоки и значения вероятностных числовых оценок, существуют разные взгляды, и между сторонниками этих взглядов ведутся нескончаемые споры. **Фреквентистская** (от слова frequency — частота) позиция состоит в том, что эти числовые оценки могут быть основаны только на экспериментах: если мы обследуем 100 человек и обнаружим, что у 10 из них в зубах имеется дупло, то лишь в этом случае можем утверждать, что веро-

ятность наличия дупла примерно равна 0,1. Согласно этим взглядам, утверждение “вероятность дупла равна 0,1” означает, что 0,1 — это доля соответствующих пациентов, которая будет наблюдаться в пределе при обследовании выборки, состоящей из бесконечного количества людей. А любая конечная выборка позволяет оценить истинную долю, а также рассчитать, насколько точной, по-видимому, является эта оценка.

Согласно **объективистским** взглядам, вероятности — это реальные аспекты универсума (связанные с тем, что в поведении объектов наблюдаются определенные закономерности), а не просто описания степени уверенности наблюдателя. Например, тот факт, что подлинная монета падает решкой вверх с вероятностью 0,5, отражает закономерности падения самой монеты. С точки зрения сторонников этих взглядов, фреквентистские измерения представляют собой попытки проведения наблюдений за реализацией таких закономерностей. Большинство физиков согласны с тем, что квантовые феномены, наблюдаемые в микроскопических масштабах, объективно являются вероятностными, а неопределенность в макроскопических масштабах (например, при подбрасывании монеты) обычно обусловлена незнанием начальных условий и, скорее всего, не согласуется с этими взглядами о реализации каких-то закономерностей.

Сторонники **субъективистских** взглядов описывают вероятности как способ описания уверенности агента, а не как проявления, имеющие какую-то внешнюю физическую значимость. Именно это позволяет врачу или аналитику оперировать с числами, например, чтобы иметь возможность утверждать: “По моему мнению, ожидаемая вероятность наличия дупла составляет около 0,1”. Кроме того, для выявления вероятностных оценок, применяемых людьми, было разработано несколько более надежных методов, таких как системы регистрации ставок, описанные на с. 637.

В конце концов, даже строго фреквентистская позиция требует субъективного анализа, поэтому с точки зрения практики различия между взглядами сторонников этих трех направлений, по-видимому, не имеют большого значения. То, что нельзя обойтись без субъективного подхода, иллюстрирует проблема **референтного класса** (reference class). Предположим, что некий врач, сторонник фреквентистских взглядов, желает знать, каковы шансы того, что некоторый пациент имеет данное конкретное заболевание. Этот врач хочет рассмотреть сведения о других пациентах, которые обладают аналогичными характеристиками по важным показателям (возраст, симптомы и, возможно, пол), и определить, какая часть из них имеет это заболевание. Такая задача может быть решена, но если врач станет рассматривать все, что известно о пациенте (вес с точностью до грамма, цвет волос, девичья фамилия матери и т.д.), результатом станет то, что не найдется больше пациентов, имеющих точно такие же характеристики, и поэтому нельзя будет определить референтный класс, в котором можно было бы собрать экспериментальные данные. Это — одна из неразрешимых проблем в философии науки.

В **принципе безразличия** Лапласа [887] утверждается, что высказывания, которые являются синтаксически “симметричными” по отношению к данному свидетельству, должны рассматриваться как равновероятные. Были предложены различные уточнения этого принципа, а кульминацией этих усилий ста-

ла попытка Карнапа и других ученых разработать строгую **индуктивную логику**, позволяющую правильно вычислять вероятность любого высказывания на основании результатов любой коллекции наблюдений. В настоящее время считается, что никакой уникальной индуктивной логики не существует; вместо этого любая подобная логика опирается на субъективное априорное распределение вероятностей, субъективность которой уменьшается по мере накопления все большего и большего количества наблюдений.

### 13.3. АКСИОМЫ ВЕРОЯТНОСТЕЙ

До сих пор в этой главе был определен синтаксис для высказываний, а также для априорных и условных вероятностных утверждений об этих высказываниях. Теперь необходимо определить своего рода семантику для вероятностных утверждений. Начнем с базовых аксиом, которые служат для определения шкалы вероятностей и ее конечных точек, как описано ниже.

1. Все вероятности находятся в пределах от 0 до 1; для любого высказывания  $a$  справедливо следующее:

$$0 \leq P(a) \leq 1$$

2. Безусловно истинные (т.е. выполнимые) высказывания имеют вероятность 1, а безусловно ложные (т.е. невыполнимые) высказывания имеют вероятность 0:

$$P(\text{true}) = 1 \quad P(\text{false}) = 0$$

Кроме того, требуется аксиома, которая соединяет вероятности логически взаимосвязанных высказываний. Такую аксиому можно проще всего составить, определив вероятность дизъюнкции, как показано ниже.

3. Вероятность дизъюнкции задается следующей формулой:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Это правило можно легко запомнить, отметив, что те случаи, когда высказывание  $a$  является истинным, вместе с теми случаями, когда  $b$  является истинным, безусловно, охватывают все те случаи, когда истинно высказывание  $a \vee b$ ; но в сумме двух множеств случаев их пересечение встречается дважды, поэтому необходимо вычесть  $P(a \wedge b)$ .

Эти три аксиомы часто называют **аксиомами Колмогорова** в честь советского математика Андрея Колмогорова, который показал, как построить остальную часть теории вероятностей на этом простом фундаменте. Обратите внимание на то, что в этих аксиомах речь идет только об априорных вероятностях, а не об условных; это связано с тем, что последние уже были определены в терминах первых в уравнении 13.1.

#### Использование аксиом вероятностей

Из этих основных аксиом можно вывести целый ряд полезных фактов. Например, знакомое правило отрицания следует из подстановки  $\neg a$  вместо  $b$  в аксиому 3, что приводит к получению следующего выражения:

$$\begin{aligned}
 P(a \vee \neg a) &= P(a) + P(\neg a) - P(a \wedge \neg a) && \text{(согласно аксиоме 3 с } b = \neg a) \\
 P(\text{true}) &= P(a) + P(\neg a) - P(\text{false}) && \text{согласно правилу логической} \\
 1 &= P(a) + P(\neg a) && \text{эквивалентности} \\
 P(\neg a) &= 1 - P(a) && \text{согласно аксиоме 2} \\
 &&& \text{(согласно алгебраическому} \\
 &&& \text{определению)}
 \end{aligned}$$

Третья строка этого логического вывода сама является полезным фактом и может быть распространена с данного булева случая на общий дискретный случай. Допустим, что дискретная переменная  $D$  имеет область определения  $\langle d_1, \dots, d_n \rangle$ . Тогда можно легко показать (упр. 13.2), что справедлива следующая формула:

$$\sum_{i=1}^n P(D=d_i) = 1$$

Это означает, что любое вероятностное распределение по одной переменной должно в сумме<sup>6</sup> составлять 1. Справедливо также утверждение, что любое совместное распределение вероятностей по любому множеству переменных должно в сумме составлять 1; в этом можно убедиться, создав одну мегапеременную, областью определения которой является перекрестное произведение областей определения первоначальных переменных.

Напомним, что любое высказывание  $a$  эквивалентно дизъюнкции всех атомарных событий, в которых  $a$  является истинным; назовем эту дизъюнкцию множеством событий  $e(a)$ . Напомним также, что атомарные события являются взаимно исключающими, поэтому вероятность любой конъюнкции атомарных событий равна нулю, согласно аксиоме 2. Таким образом, из аксиомы 3 можно вывести следующее простое соотношение: ~~если~~ вероятность любого высказывания равна сумме вероятностей атомарных событий, в которых оно является истинным; т.е. вывести такое уравнение:

$$P(a) = \sum_{e_i \in e(a)} P(e_i) \quad (13.2)$$

Это уравнение предоставляет простой метод вычисления вероятности любого высказывания при наличии полного совместного распределения, которое задает вероятности всех атомарных событий (см. раздел 13.4.) В следующих разделах будут выведены дополнительные правила для манипулирования вероятностями. Но вначале исследуем теоретические основы самих этих аксиом.

### Теоретическое обоснование аксиом вероятностей

Аксиомы вероятностей могут рассматриваться как ограничивающие множество вероятностных убеждений, которых может придерживаться некоторый агент. Такой подход в определенной степени подобен логическому подходу, согласно которому,

---

<sup>6</sup> Для непрерывных переменных эта сумма заменяется интегралом:  $\int_{-\infty}^{\infty} P(X = x) dx = 1$ .

например, логический агент не может одновременно быть уверенном в истинности высказываний  $A$ ,  $B$  и  $\neg(A \wedge B)$ . Тем не менее в вероятностном подходе возникает дополнительное усложнение. В логическом случае семантическое определение конъюнкции означает, что, по меньшей мере, одно из трех только что упомянутых убеждений должно быть ложным в этом мире, поэтому для любого агента неразумно было бы сохранять уверенность в истинности этих трех высказываний. Но в случае вероятностей, с другой стороны, утверждения относятся не непосредственно к самому миру, а к собственному состоянию знаний агента. В таком случае, почему агент не мог бы иметь приведенное ниже множество убеждений, которое явно нарушает аксиому 3?

$$\begin{aligned} P(a) &= 0.4 \\ P(a \wedge b) &= 0.0 \\ P(b) &= 0.3 \\ P(a \vee b) &= 0.8 \end{aligned} \tag{13.3}$$

Такого рода вопрос был предметом яростных дебатов, продолжавшихся в течение нескольких десятилетий, между теми, кто отстаивал допустимость использования вероятностей как единственной обоснованной формы оценки степеней уверенности, и теми, кто отстаивал альтернативные подходы. В данном разделе мы приведем один довод в пользу аксиом вероятностей, впервые сформулированный в 1931 году в работах Бруно де Финетти (Bruno de Finetti).

Ключом к этому доводу де Финетти является связь между степенью уверенности и действиями. Идея состоит в том, что если агент имеет некоторую степень уверенности в истинности высказывания  $a$ , то агент должен быть способен сформулировать оценку того, в какой степени он безразличен к ставке за или против высказывания  $a$ . Рассмотрим эту ситуацию как игру между двумя агентами: агент 1 утверждает “моя степень уверенности в истинности события  $a$  равна 0,4”. Затем агент 2 вправе выбрать, будет ли он делать ставку за или против высказывания  $a$ , применяя суммы, которые совместимы с заявленной степенью уверенности. Это означает, что агент 2 может решить сделать ставку на то, что событие  $a$  произойдет, поставив 4 доллара против 6 долларов агента 1. С другой стороны, агент 2 может поставить 6 долларов против 4, что  $A$  не произойдет<sup>7</sup>. Если степень уверенности агента не точно отражает состояние мира, можно вполне рассчитывать на то, что в долговременной перспективе он будет проигрывать деньги агенту-противнику, убеждения которого более точно отражают состояние мира.

Но де Финетти доказал гораздо более сильное утверждение: *если агент 1 руководствуется множеством степеней уверенности, которое нарушает аксиомы теории вероятностей, то существует такая комбинация ставок агента 2, которая гарантирует, что агент 1 будет терять деньги при каждой ставке*. Поэтому, если вы придерживаетесь той идеи, что агент должен стремиться “ставить свои деньги на те события, вероятности которых выше”, то должны также признать, что нерационально иметь такие убеждения, которые нарушают аксиомы вероятностей.

<sup>7</sup> Можно было бы возразить, что предпочтения агента применительно к балансам разных ставок являются таковыми, что возможность потерять 1 доллар не уравновешивается равной возможностью выиграть 1 доллар. Один из возможных ответов на это возражение состоит в том, что суммы ставок должны быть достаточно малыми для того, чтобы можно было избежать данной проблемы. Анализ, проведенный Сэвежем [1354], позволяет полностью исключить из рассмотрения эту проблему.

На первый взгляд может показаться, что эта игра со ставками довольно надумана. Например, что было бы, если один из агентов отказался бы делать ставки? Закончился бы на этом спор? Ответ на данный вопрос состоит в том, что эта игра со ставками представляет собой абстрактную модель для ситуации принятия решений, в которую любой агент неизбежно вовлечен в любой момент своего существования. Каждое действие (включая бездействие) — это своего рода ставка, а каждый результат может рассматриваться как положительное или отрицательное вознаграждение за эту ставку. Отказ делать ставку аналогичен решению предоставить все естественному течению событий.

Мы не будем доказывать представленную выше теорему де Финетти, а приведем один пример. Предположим, что агент 1 руководствуется множеством степеней уверенности, приведенным в уравнении 13.3. В табл. 13.1 показано, что если агент 2 решит делать ставки 4 доллара на событие  $a$ , 3 доллара — на событие  $b$  и 2 доллара на событие  $\neg(a \vee b)$ , то агент 1 будет всегда терять деньги, независимо от результатов  $a$  и  $b$ .

**Таблица 13.1. Пример того, что при наличии у агента 1 несогласованных убеждений агент 2 получает возможности составить множество ставок, которое гарантирует потери для агента 1, независимо от результатов  $a$  и  $b$**

Агент 1		Агент 2		Результат для Агента 1			
Высказывание	Степень уверенности	Ставка	Сумма ставки	$a \wedge b$	$a \wedge \neg b$	$\neg a \wedge b$	$\neg a \wedge \neg b$
$a$	0.4	$a$	4–6	-6	-6	4	4
$b$	0.3	$b$	3–7	-7	3	-7	3
$a \vee b$	0.8	$\neg(a \vee b)$	2–8	2	2	2	-8
				-11	-1	-1	-1

В пользу применения вероятностей были выдвинуты и другие весомые философские аргументы; к числу наиболее широко известных из них относятся сформулированные Коксом [304] и Карнапом [225]. Но мир таков, каким он является, и практические свидетельства иногда становятся более весомыми, чем доказательства. Успех систем формирования рассуждений, основанных на теории вероятностей, оказался гораздо более привлекательным стимулом, который вызвал пересмотр многих взглядов. В следующем разделе будет показано, как описанные выше аксиомы могут быть применены для составления логических выводов.

## 13.4. ЛОГИЧЕСКИЙ ВЫВОД С ИСПОЛЬЗОВАНИЕМ ПОЛНЫХ СОВМЕСТНЫХ РАСПРЕДЕЛЕНИЙ

В данном разделе будет описан простой метод **вероятностного вывода**, т.е. вычисления апостериорных вероятностей для высказываний, заданных в виде запросов, на основании наблюдаемых свидетельств. Мы будем использовать полное совместное распределение как своего рода “базу знаний”, из которой могут быть выведены ответы на все вопросы. В ходе этого мы также представим несколько

полезных методов манипулирования уравнениями, в которых учитываются вероятности.

Начнем с очень простого примера — с описания проблемной области, состоящей только из трех булевых переменных, *Toothache*, *Cavity* и *Catch* (неприятные ощущения от захвата зуба стальными клещами дантиста все еще свежи в памяти автора). Полное совместное распределение представляет собой таблицу с размерами  $2 \times 2 \times 2$  (табл. 13.2).

**Таблица 13.2. Полное совместное распределение для мира *Toothache*, *Cavity*, *Catch***

		<i>toothache</i>		$\neg$ <i>toothache</i>	
		<i>catch</i>	$\neg$ <i>catch</i>	<i>catch</i>	$\neg$ <i>catch</i>
<i>cavity</i>	<i>catch</i>	0.108	0.012	0.072	0.008
	$\neg$ <i>catch</i>	0.016	0.064	0.144	0.576

Обратите внимание на то, что вероятности в этом совместном распределении в сумме составляют 1, как и требуется согласно аксиомам вероятностей. Следует также отметить, что уравнение 13.2 предоставляет нам прямой способ вычисления вероятности любого высказывания, простого или сложного: мы должны определить те атомарные события, в которых данное высказывание является истинным, и сложить их вероятности. Например, имеется шесть атомарных событий, в которых истинным является высказывание *cavity*  $\vee$  *toothache*:

$$P(\text{cavity} \vee \text{toothache}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$$

Одна из задач, которая встречается особенно часто, состоит в том, чтобы извлечь из подобной таблицы распределение вероятностей по некоторому подмножеству переменных или по одной переменной. Например, складывая элементы первого ряда табл. 13.2, получим безусловную, или **маргинальную**, вероятность<sup>8</sup> события *cavity*:

$$P(\text{cavity}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

Такой процесс называется **маргинализацией**, или **исключением из суммы**, поскольку из суммы вероятностей исключаются прочие переменные, кроме *Cavity*. Можно записать следующее общее правило маргинализации для любых множеств переменных **Y** и **Z**:

$$P(\mathbf{Y}) = \sum_{\mathbf{Z}} P(\mathbf{Y}, \mathbf{Z}) \quad (13.4)$$

Это означает, что распределение вероятностей по **Y** может быть получено путем исключения из суммы вероятностей всех прочих переменных, относящихся к любому совместному распределению вероятностей, содержащему **Y**. В одном из вариантов этого правила учитываются условные вероятности, а не совместные вероятности с использованием правила произведения:

<sup>8</sup> Эта вероятность получила такое название, поскольку страховщики имеют общую привычку записывать суммы наблюдаемых частот событий на полях (*margin*) таблиц страхования.

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}|\mathbf{z}) P(\mathbf{z}) \quad (13.5)$$

Это правило называется правилом **обусловливания**. Как оказалось, правила маргинализации и обусловливания являются очень полезными правилами для всех видов логических выводов, в которых применяются вероятностные выражения.

В большинстве случаев нас будет интересовать задача вычисления условных вероятностей некоторых переменных при наличии свидетельств, касающихся других переменных. Условные вероятности можно найти, вначале воспользовавшись уравнением 13.1 для получения выражения в терминах безусловных вероятностей, а затем рассчитав это выражение на основании полного совместного распределения. Например, можно вычислить вероятность наличия дупла после получения свидетельства о том, что пациент страдает от зубной боли, следующим образом:

$$\begin{aligned} \mathbf{P}(\text{cavity} | \text{toothache}) &= \frac{\mathbf{P}(\text{cavity} \wedge \text{toothache})}{\mathbf{P}(\text{toothache})} \\ &= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6 \end{aligned}$$

Просто для проверки мы можем также рассчитать вероятность того, что у пациента нет дупла, если у него наблюдается зубная боль:

$$\begin{aligned} \mathbf{P}(\neg\text{cavity} | \text{toothache}) &= \frac{\mathbf{P}(\neg\text{cavity} \wedge \text{toothache})}{\mathbf{P}(\text{toothache})} \\ &= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4 \end{aligned}$$

Обратите внимание на то, что в этих двух примерах вычисления вероятности терм  $1/\mathbf{P}(\text{toothache})$  остается постоянным, независимо от того, какое значение *Cavity* вычисляется. Фактически этот терм может рассматриваться как константа **нормализации** для распределения  $\mathbf{P}(\text{Cavity} | \text{toothache})$ , гарантирующая, что полученные вероятности в сумме составят 1. Во всех главах этой книги, где речь идет о вероятностях, для обозначения подобных констант будем использовать символ  $\alpha$ . С помощью такого обозначения можно записать два приведенных выше уравнения в виде одного:

$$\begin{aligned} \mathbf{P}(\text{Cavity} | \text{toothache}) &= \alpha \mathbf{P}(\text{Cavity}, \text{toothache}) \\ &= \alpha [\mathbf{P}(\text{Cavity}, \text{toothache}, \text{catch}) + \mathbf{P}(\text{Cavity}, \text{toothache}, \neg\text{catch})] \\ &= \alpha [<0.108, 0.016> + <0.012, 0.064>] = \alpha <0.12, 0.08> = <0.6, 0.4> \end{aligned}$$

Как оказалось, нормализация является полезным сокращением во многих вероятностных вычислениях.

На основании приведенного выше примера можно составить общую процедуру вероятностного вывода. Мы будем придерживаться того случая, в котором запрос касается только одной переменной. Кроме того, потребуются некоторые обозначения: допустим, что  $X$  — переменная запроса (в данном примере *Cavity*);  $\mathbf{E}$  — множество переменных свидетельства (в данном примере к нему относится только *Toothache*);  $\mathbf{e}$  — наблюдаемые значения этих переменных;  $\mathbf{Y}$  — оставшиеся ненаблюдаемые переменные (в данном примере таковой является только *Catch*). Запросом является  $\mathbf{P}(X | \mathbf{e})$ , и ответ на него может быть вычислен следующим образом:

$$\mathbf{P}(X|\mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) \quad (13.6)$$

где суммирование осуществляется по всем возможным значениям  $\mathbf{y}$  (т.е. по всем возможным комбинациям значений ненаблюдаемых переменных  $\mathbf{Y}$ ). Обратите внимание на то, что переменные  $X$ ,  $\mathbf{E}$  и  $\mathbf{Y}$ , вместе взятые, составляют полное множество переменных для данной проблемной области, поэтому  $\mathbf{P}(X, \mathbf{e}, \mathbf{y})$  представляет собой подмножество вероятностей из полного совместного распределения. Алгоритм вычисления ответа на запрос приведен в листинге 13.2. В этом алгоритме проверяются в циклах значения  $X$  и значения  $Y$  для перебора всех возможных атомарных событий с фиксированными значениями  $\mathbf{e}$ , складываются их вероятности из таблицы совместного распределения, после чего результаты нормализуются.

#### Листинг 13.2. Алгоритм вероятностного вывода путем перебора всех элементов в таблице полного совместного распределения

---

```

function Enumerate-Joint-Ask( $X$ ,  $\mathbf{e}$ ,  $\mathbf{P}$ ) returns распределение по  $X$ 
  inputs:  $X$ , переменная запроса
            $\mathbf{e}$ , наблюдаемые значения переменных  $\mathbf{E}$ 
            $\mathbf{P}$ , совместное распределение по переменным
            $\{X\} \cup \mathbf{E} \cup \mathbf{Y}\}$  /*  $\mathbf{Y}$  - скрытые переменные */

   $\mathbf{Q}(X) \leftarrow$  распределение по  $X$ , первоначально пустое
  for each значение  $x_i$  переменной  $X$  do
     $\mathbf{Q}(x_i) \leftarrow$  Enumerate-Joint( $x_i$ ,  $\mathbf{e}$ ,  $\mathbf{Y}$ , [],  $\mathbf{P}$ )
  return Normalize( $\mathbf{Q}(X)$ )

function Enumerate-Joint( $x$ ,  $\mathbf{e}$ ,  $vars$ ,  $values$ ,  $\mathbf{P}$ ) returns действительное
  число
  if Empty?( $vars$ ) then return  $\mathbf{P}(x, \mathbf{e}, values)$ 
   $Y \leftarrow$  First( $vars$ )
  return  $\sum_y$  Enumerate-Joint( $x$ ,  $\mathbf{e}$ , Rest( $vars$ ), [ $y | values$ ],  $\mathbf{P}$ )

```

---

При наличии полного совместного распределения, с которым можно было бы работать, алгоритм Enumerate-Joint-Ask становится полным алгоритмом получения ответов на вероятностные запросы, касающиеся дискретных переменных. Но этот алгоритм недостаточно хорошо масштабируется, поскольку при наличии проблемной области, которая описана  $n$  булевыми переменными, он требует входной таблицы с размером  $O(2^n)$ , а обработка этой таблицы занимает время  $O(2^n)$ . В реальных задачах может потребоваться рассмотреть сотни или тысячи случайных переменных, а не только три. Попытка определить огромные количества требуемых вероятностей быстро становится полностью неосуществимой, поскольку может быть даже не накоплено достаточного количества экспериментальных данных, которые требуются для того, чтобы отдельно оценить каждую из записей этой таблицы.

По этим причинам полное совместное распределение вероятностей в табличной форме нельзя считать практически применимым инструментальным средством для создания систем формирования рассуждений (хотя в исторических заметках в конце данной главы упоминается одно реальное приложение, основанное на этом методе).

Вместо этого данный подход следует рассматривать как теоретическую основу, на которой могут быть созданы более эффективные подходы. В оставшейся части данной главы представлены некоторые основные идеи, которые потребуются для подготовки к разработке реально осуществимых систем, описанных в главе 14.

### 13.5. НЕЗАВИСИМОСТЬ

Расширим полное совместное распределение, приведенное в табл. 13.2, введя четвертую переменную, *Weather*. После этого полное совместное распределение, состоящее из 32 элементов (поскольку переменная *Weather* имеет четыре значения), принимает вид  $P(Toothache, Catch, Cavity, Weather)$ . Оно содержит четыре варианта таблицы, показанной в табл. 13.2, по одному на каждый вид погоды. Напрашивается резонный вопрос о том, какую связь имеют эти варианты друг с другом и с первоначальной таблицей, состоящей из трех переменных. Например, как связаны друг с другом высказывания  $P(toothache, catch, cavity, Weather=cloudy)$  и  $P(toothache, catch, cavity)$ ? Один из способов получения ответа на этот вопрос состоит в использования правила произведения:

$$\begin{aligned} P(toothache, catch, cavity, Weather = \text{cloudy}) \\ = P(Weather=\text{cloudy} | toothache, catch, cavity) \\ P(toothache, catch, cavity) \end{aligned}$$

Но человек, не верящий в существование колдунов, не может себе представить, что чьи-то проблемы с зубами могут влиять на погоду. Поэтому кажется резонным следующее утверждение:

$$\begin{aligned} P(Weather = \text{cloudy} | toothache, catch, cavity) \\ = P(Weather = \text{cloudy}) \end{aligned} \quad (13.7)$$

На основании этого можно вывести следующее:

$$\begin{aligned} P(toothache, catch, cavity, Weather = \text{cloudy}) \\ = P(Weather = \text{cloudy}) P(toothache, catch, cavity) \end{aligned}$$

Аналогичное уравнение существует для каждого элемента в распределении  $P(Toothache, Catch, Cavity, Weather)$ . В действительности можно записать такое общее уравнение:

$$\begin{aligned} P(Toothache, Catch, Cavity, Weather) = \\ P(Toothache, Catch, Cavity) P(Weather) \end{aligned}$$

Таким образом, 32-элементная таблица для четырех переменных может быть сконструирована из одной 8-элементной таблицы и одной 4-элементной. Такая декомпозиция показана схематически на рис. 13.1, а.

Свойство вероятностей, используемое при составлении уравнения 13.7, называется **независимостью** (а также **маргинальной независимостью** и **абсолютной независимостью**). В частности, погода независима от чьих-то проблем с зубами. Независимость между высказываниями  $a$  и  $b$  может быть показана следующим образом:

$$P(a | b) = P(a) \text{ или } P(b | a) = P(b) \text{ или } P(a \wedge b) = P(a)P(b) \quad (13.8)$$

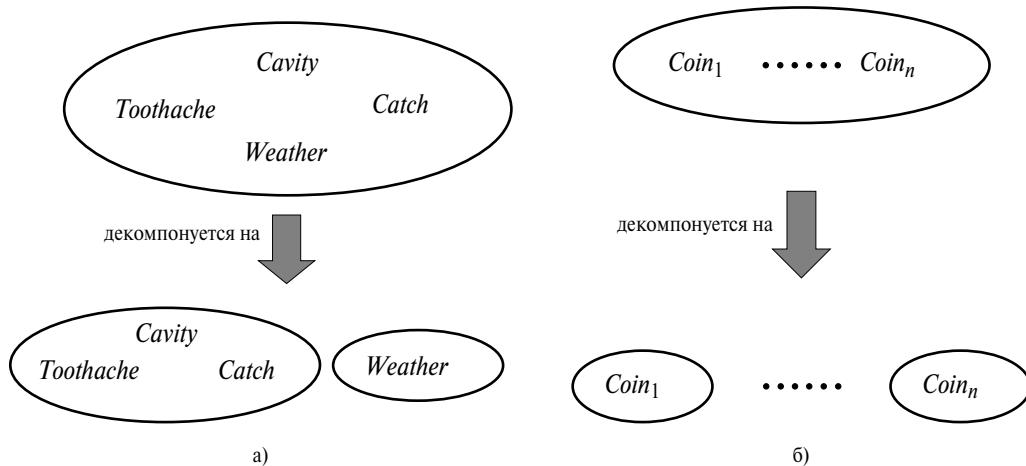


Рис. 13.1. Два примера факторизации большого совместного распределения на меньшие распределения с использованием свойства абсолютной независимости: независимы друг от друга погода и проблемы с зубами (а); независимы друг от друга броски монеты (б)

Все эти формы записи являются эквивалентными (упр. 13.7). Свойство независимости между переменными  $X$  и  $Y$  можно сформулировать следующим образом (все эти формы записи также эквивалентны):

$$\mathbf{P}(X|Y) = \mathbf{P}(X) \text{ или } \mathbf{P}(Y|X) = \mathbf{P}(Y) \text{ или } \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y)$$

Утверждения о независимости обычно основаны на знаниях о проблемной области. Как показывает приведенный выше пример, эти утверждения позволяют существенно уменьшить объем информации, необходимой для описания полного совместного распределения. Если все множество переменных может быть разделено на независимые подмножества, то полное совместное распределение может быть факторизовано на отдельные совместные распределения, заданные на этих подмножествах. Например, совместное распределение результатов  $n$  независимых бросков монеты,  $\mathbf{P}(C_1, \dots, C_n)$ , может быть представлено как произведение  $n$  распределений  $\mathbf{P}(C_i)$  с одной переменной. Например, с точки зрения практики очень благоприятным фактором является независимость данных зубоврачебного дела и метеорологии, поскольку в противном случае для занятия зубоврачебным делом потребовались бы глубокие знания в области метеорологии, и наоборот.

Поэтому утверждения о независимости, если они имеются, позволяют сократить размеры представления проблемной области и уменьшить сложность проблемы вывода. К сожалению, чистое разделение целых множеств переменных по признаку независимости встречается редко. Если между двумя переменными существует хоть какая-то связь, пусть даже косвенная, свойство независимости перестает соблюдать-ся. Кроме того, даже независимые подмножества могут оказаться чрезвычайно большими, например, в области стоматологии могут встретиться десятки заболеваний и сотни симптомов, причем все они взаимосвязаны друг с другом. Чтобы справиться с такими проблемами, мы должны иметь более тонкие методы, чем прямолинейная концепция независимости.

## 13.6. ПРАВИЛО БАЙЕСА И ЕГО ИСПОЛЬЗОВАНИЕ

На с. 632 определено **правило произведения** и указано, что оно может быть записано в двух следующих формах благодаря коммутативности конъюнкции:

$$\begin{aligned} P(a \wedge b) &= P(a|b)P(b) \\ P(a \wedge b) &= P(b|a)P(a) \end{aligned}$$

Приравняв две правые части стороны и разделив их на  $P(a)$ , получим такое уравнение:

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} \quad (13.9)$$

Это уравнение известно под названием **правила Байеса** (а также *закона Байеса*, или *теоремы Байеса*)<sup>9</sup>. Это простое уравнение лежит в основе всех современных систем искусственного интеллекта для вероятностного вывода. Более общий случай многозначных переменных может быть записан в системе обозначений **P** следующим образом:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Это уравнение также следует рассматривать как представляющее множество уравнений, в каждом из которых рассматриваются конкретные значения переменных. Время от времени нам также придется использовать более общую версию, которая обусловлена некоторым фоновым свидетельством **e**:

$$P(Y|X, e) = \frac{P(X|Y, e)P(Y|e)}{P(X|e)} \quad (13.10)$$

### Применение правила Байеса: простой случай

На первый взгляд правило Байеса не кажется очень полезным. В нем требуются три терма (одна условная вероятность и две безусловных вероятности) только для вычисления одной условной вероятности.

Но правило Байеса находит очень широкое практическое применение, поскольку во многих случаях имеются хорошие оценки вероятностей для этих трех термов и нужно вычислить четвертый. В такой задаче, как медицинская диагностика, часто известны условные вероятности причинных связей и требуется определить диагноз. Врач знает, что такое заболевание, как менингит, очень часто вызывает у пациента симптом, характеризующийся снижением подвижности шеи; предположим, что этот симптом наблюдается в 50% случаев. Кроме того, врачу известны некоторые безусловные факты: априорная вероятность того, что некоторый пациент имеет менингит, равна 1/50 000, а априорная вероятность того, что некоторый пациент имеет неподвижную шею, равна 1/20. Предположив, что *s* — высказывание, согласно которому пациент имеет неподвижную шею, а *m* — высказывание, что пациент имеет менингит, получим следующее:

<sup>9</sup> Согласно правилу 1, приведенному на странице 1 книги *The Elements of Style* (Основы стилистики) Шранка и Уайта, в англоязычном эквиваленте этого термина следует применять форму родительного падежа Bayes's, а не Bayes'. Однако чаще используется последняя форма.

$$\begin{aligned}
 P(s|m) &= 0.5 \\
 P(m) &= 1/50000 \\
 P(s) &= 1/20 \\
 P(m|s) &= \frac{P(s|m)P(m)}{P(s)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002
 \end{aligned}$$

Итак, следует предполагать, что 1 из 5000 пациентов с неподвижной шеей имеет менингит. Следует отметить, что даже если неподвижная шея является весьма надежным показателем наличия менингита (с вероятностью 0,5), сама вероятность наличия менингита у пациента остается низкой. Это связано с тем, что априорная вероятность наличия симптома неподвижной шеи намного выше по сравнению с вероятностью менингита.

В разделе 13.4 показан процесс, с помощью которого можно избежать необходимости оценки вероятности свидетельства (в данном случае  $P(s)$ ), вместо этого вычислив апостериорную вероятность для каждого значения переменной запроса (в данном случае  $m$  и  $\neg m$ ), а затем нормализовав результаты. Тот же процесс можно применить при использовании правила Байеса. Таким образом, мы имеем:

$$\mathbf{P}(M|s) = \alpha \langle P(s|m)P(m), P(s|\neg m)P(\neg m) \rangle$$

Итак, чтобы воспользоваться этим подходом, необходимо вместо  $P(s)$  вычислить значение  $P(s|\neg m)$ . Осуществление такого подхода требует определенных затрат; иногда эти затраты не столь велики, а иногда становятся довольно значительными. Общая форма правила Байеса с нормализацией является таковой:

$$\mathbf{P}(Y|X) = \alpha \mathbf{P}(X|Y) \mathbf{P}(Y) \quad (13.11)$$

где  $\alpha$  — константа нормализации, необходимая для того, чтобы записи в распределении  $\mathbf{P}(Y|X)$  в сумме составляли 1.

Один из очевидных вопросов, касающихся правила Байеса, состоит в том, почему может оказаться доступной условная вероятность, реализуемая в одном направлении, но не в другом. В проблемной области лечения менингита, возможно, врач знает, что из симптома неподвижной шеи следует наличие менингита в 1 из 5000 случаев; это означает, что врач имеет количественную информацию в **диагностическом** направлении, от симптомов к причинам. Для такого врача не требуется использование правила Байеса. К сожалению, ~~диагностические знания часто встречаются~~ *намного реже по сравнению с причинными знаниями*. Если внезапно возникает эпидемия менингита, то безусловная вероятность менингита,  $P(m)$ , повышается. Врач, который вывел диагностическую вероятность  $P(m|s)$  непосредственно из статистических наблюдений за пациентами перед эпидемией, не будет иметь представления о том, как обновить это значение, а врач, который вычисляет  $P(m|s)$  из других трех значений, обнаружит, что значение  $P(m|s)$  должно увеличиваться пропорционально  $P(m)$ . Еще более важно то, что причинная информация  $P(s|m)$  остается незатронутой данной эпидемией, поскольку она просто показывает, в чем выражается действие менингита. Использование такого рода прямых причинных знаний, или знаний, основанных на модели, позволяет достичь надежности, которая крайне важна при создании вероятностных систем, применимых в реальном мире.

## Использование правила Байеса: комбинирование свидетельств

Как было показано выше, правило Байеса может применяться для получения ответов на вероятностные запросы, в которых учтено условие, составляющее одно из свидетельств, например неподвижная шея. В частности, было показано, что вероятностная информация часто доступна в форме  $P(effect|cause)$  (где *effect* — результат, а *cause* — причина). А что произойдет, если свидетельств два или больше? Например, какой вывод может сделать зубной врач, если его стальной инструмент захватил больной зуб пациента, причинив еще большие страдания? Если известно полное совместное распределение (табл. 13.2), можно сразу же прочитать ответ:

$$P(Cavity|toothache \wedge catch) = \alpha <0.108, 0.016> \approx <0.871, 0.129>$$

Но нам уже известно, что такой подход не масштабируется на большее количество переменных.

Тогда можно попытаться воспользоваться правилом Байеса для переформулировки этой задачи:

$$\begin{aligned} P(Cavity|toothache \wedge catch) &= \\ \alpha P(toothache \wedge catch|Cavity) P(Cavity) &\end{aligned} \quad (13.12)$$

Для того чтобы можно было найти ответ запрос в такой формулировке, необходимо знать условные вероятности конъюнкции  $toothache \wedge catch$  для каждого значения *Cavity*. Такая задача может быть осуществима, если речь идет только о двух переменных свидетельства, но этот подход снова становится источником затруднений при его применении в больших масштабах. Если имеется  $n$  возможных переменных свидетельства (рентгеновское обследование, диета, гигиена полости рта и т.д.), то количество возможных комбинаций наблюдаемых значений, для которых необходимо будет знать условные вероятности, составит  $2^n$ . С таким же успехом можно было бы снова вернуться к использованию полного совместного распределения. Именно по этой причине исследователи после первых попыток отказались от применения теории вероятностей и обратились к приближенным методам комбинирования свидетельств, в которых требуется использовать меньше чисел для получения ответов, хотя сами эти ответы не всегда бывают правильными.

Вместо того чтобы следовать по такому пути, мы должны найти некоторые дополнительные утверждения о рассматриваемой проблемной области, которые позволят упростить применяемые выражения. Понятие **независимости**, приведенное в разделе 13.5, дает ключ к этому решению, но требует уточнения. Было бы прекрасно, если бы переменные *Toothache* и *Catch* были независимыми, но они таковыми не являются: если зубной врач захватывает зуб своим инструментом, то он делает это, вероятно, потому, что в этом зube есть дупло, а это действие, вероятно, в свою очередь вызывает боль. Но эти переменные независимы, если речь идет о наличии или отсутствии дупла. Причиной каждого из соответствующих действий было дупло, но ни одно из них не оказывает непосредственного влияния на другое: зубная боль зависит от состояния нервов в зube, а точность наложения инструмента зависит от навыков зубного врача<sup>10</sup>, к которым зубная боль не имеет отношения. Математически это свойство записывается следующим образом:

<sup>10</sup> Предполагается, что пациент и зубной врач — это разные люди.

$$\begin{aligned} \mathbf{P}(\text{toothache} \wedge \text{catch} | \text{Cavity}) &= \\ \mathbf{P}(\text{toothache} | \text{Cavity}) \mathbf{P}(\text{catch} | \text{Cavity}) \end{aligned} \quad (13.13)$$

В данном уравнении выражена **условная независимость** переменных *toothache* и *catch*, если дана вероятность *Cavity*. Соответствующее выражение можно вставить в уравнение 13.12 для определения вероятности наличия дупла:

$$\begin{aligned} \mathbf{P}(\text{Cavity} | \text{toothache} \wedge \text{catch}) &= \\ \alpha \mathbf{P}(\text{toothache} | \text{Cavity}) \mathbf{P}(\text{catch} | \text{Cavity}) \mathbf{P}(\text{Cavity}) \end{aligned}$$

Теперь требования к наличию информации становятся такими же, как и при вероятностном выводе с использованием каждого свидетельства отдельно: необходимо знать априорную вероятность  $\mathbf{P}(\text{Cavity})$  для переменной запроса и условную вероятность каждого результата, если дана его причина.

Общее определение условной независимости двух переменных *X* и *Y*, если дана третья переменная *Z*, выражается следующей формулой:

$$\mathbf{P}(X, Y | Z) = \mathbf{P}(X | Z) \mathbf{P}(Y | Z)$$

Например, в проблемной области стоматологии представляется вполне резонным применение утверждения об условной независимости переменных *Toothache* и *Catch*, если дана вероятность *Cavity*:

$$\begin{aligned} \mathbf{P}(\text{Toothache}, \text{Catch} | \text{Cavity}) &= \\ \mathbf{P}(\text{Toothache} | \text{Cavity}) \mathbf{P}(\text{Catch} | \text{Cavity}) \end{aligned} \quad (13.14)$$

Обратите внимание на то, что это утверждение немного строже по сравнению с уравнением 13.13, в котором сформулировано утверждение о независимости только для конкретных значений *Toothache* и *Catch*. А при использовании свойства абсолютной независимости, сформулированного в уравнении 13.8, могут также применяться следующие эквивалентные формы:

$$\mathbf{P}(X | Y, Z) = \mathbf{P}(X | Z) \text{ и } \mathbf{P}(Y | X, Z) = \mathbf{P}(Y | Z)$$

В разделе 13.5 показано, что утверждения с описанием свойств абсолютной независимости позволяют выполнять декомпозицию полного совместного распределения на гораздо более мелкие распределения. Как оказалось, аналогичную декомпозицию позволяют выполнять утверждения об условной независимости. Например, с помощью утверждения, приведенного в уравнении 13.14, декомпозицию можно вывести следующим образом:

$$\begin{aligned} \mathbf{P}(\text{Toothache}, \text{Catch}, \text{Cavity}) & \\ = \mathbf{P}(\text{Toothache}, \text{Catch} | \text{Cavity}) \mathbf{P}(\text{Cavity}) & \text{(по правилу} \\ & \text{произведения)} \\ = \mathbf{P}(\text{Toothache} | \text{Cavity}) \mathbf{P}(\text{Catch} | \text{Cavity}) \mathbf{P}(\text{Cavity}) & \text{(согласно} \\ & \text{уравнению 13.14)} \end{aligned}$$

Таким образом, первоначальная крупная таблица декомпонована на три меньшие таблицы. В исходной таблице было семь независимых чисел ( $2^3 - 1$ , поскольку эти числа должны в сумме составлять 1). Меньшие таблицы содержат пять независимых чисел ( $2 \times (2^1 - 1)$  для каждого распределения условных вероятностей и  $2^1 - 1$  для распределения априорной вероятности *Cavity*). Такое достижение на первый взгляд может показаться не очень значительным, но дело в том, что для *n* симптомов, являющихся условно независимыми, если дана вероятность *Cavity*, размер представления растет как  $O(n)$ , а не  $O(2^n)$ . Таким об-

разом, ~~если~~ утверждения об условной независимости могут обеспечивать масштабирование вероятностных систем; более того, такие утверждения могут быть подкреплены данными намного проще по сравнению с утверждениями об абсолютной независимости. С концептуальной точки зрения переменная *Cavity* ~~и~~ разделяет переменные *Toothache* и *Catch*, поскольку наличие дупла является прямой причиной и зубной боли, и наложения инструмента на зуб. Разработка методов декомпозиции крупных вероятностных областей определения на слабо связанные подмножества с помощью свойства условной независимости стало одним из наиболее важных достижений в новейшей истории искусственного интеллекта.

Приведенный выше пример из области стоматологии может служить проявлением часто встречающейся ситуации, в которой одна причина непосредственно влияет на целый ряд результатов, причем все эти результаты являются условно независимыми, если дана эта причина. Полное совместное распределение может быть записано следующим образом:

$$P(Cause, Effect_1, \dots, Effect_n) = P(Cause) \prod_i P(Effect_i | Cause)$$

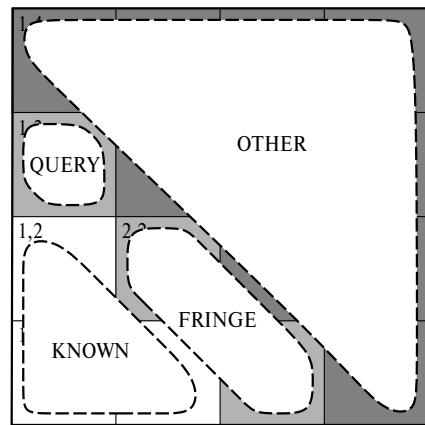
Указанное распределение вероятностей называется ~~и~~ **наивной байесовской** моделью. Такая модель называется “наивной”, поскольку часто используется (как упрощающее допущение) в тех случаях, когда переменные “результата” не являются условно независимыми, если дана переменная причины. (Наивную байесовскую модель иногда называют **байесовским классификатором**, а это не совсем корректное применение термина побудило настоящих специалистов в области байесовских моделей называть ее не наивной, а ~~и~~ **идиотской байесовской** моделью.) На практике наивные байесовские системы могут действовать удивительно успешно, даже если предположение о независимости не является истинным. В главе 20 описаны методы изучения наивных байесовских распределений по данным наблюдений.

### 13.7. ЕЩЕ ОДНО ВОЗВРАЩЕНИЕ В МИР ВАМПУСА

Теперь мы можем применить многие идеи, изложенные в этой главе, для решения задач в мире вампуса, требующих использования вероятностных рассуждений (полное описание мира вампуса приведено в главе 7). Неопределенность в мире вампуса возникает из-за того, что датчики агента сообщают только частичную, локальную информацию об этом мире. Например, на рис. 13.2 показана ситуация, в которой каждый из трех достижимых квадратов ( $[1, 3]$ ,  $[2, 2]$  и  $[3, 1]$ ) может содержать яму. Чисто логический вывод не позволяет прийти к каким-либо заключениям о том, какой квадрат с наибольшей вероятностью окажется безопасным, поэтому логический агент может быть вынужден выбирать среди них случайным образом. В этом разделе будет показано, что вероятностный агент может действовать гораздо успешнее, чем логический агент.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 B OK	2,2	3,2	4,2
1,1 OK	2,1 B OK	3,1	4,1

a)



б)

Рис. 13.2. Пример из мира вампира: после обнаружения ветерка и в квадрате [1, 2], и в квадрате [2, 1] агент заходит в тупик — больше нет такого квадрата, который он мог бы обследовать без опасений (а); разделение квадратов на категории Known (Известные), Fringe (Периферийные) и Other (Другие) для формирования запроса (Query), касающегося квадрата [1, 3] (б).

Наша цель будет состоять в том, чтобы вычислить вероятность наличия ямы в каждом из этих трех квадратов. (В данном конкретном примере игнорируется наличие вампира и золота.) Относящиеся к этой задаче свойства мира вампира состоят в том, что, во-первых, яма становится причиной возникновения ветерка во всех соседних квадратах, и, во-вторых, каждый квадрат, отличный от [1, 1], содержит яму с вероятностью 0,2. Первый этап состоит в определении множества необходимых случайных переменных, как показано ниже.

- Как и в случае пропозициональной логики, требуется по одной булевой переменной  $P_{ij}$  для каждого квадрата, которая принимает истинное значение тогда и только тогда, когда квадрат  $[i, j]$  действительно содержит яму.
- Требуются также булевые переменные  $B_{ij}$ , принимающие истинное значение тогда и только тогда, когда в квадрате  $[i, j]$  чувствуется ветерок; мы предусматриваем эти переменные только для наблюдаемых квадратов, в данном случае [1, 1], [1, 2] и [2, 1].

Следующий этап состоит в определении полного совместного распределения  $\mathbf{P}(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$ . Применяя правило произведения, получаем следующее:

$$\begin{aligned} \mathbf{P}(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) &= \\ \mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1}, \dots, P_{4,4}) \mathbf{P}(P_{1,1}, \dots, P_{4,4}) \end{aligned}$$

Эта декомпозиция позволяет очень легко определить, какими должны быть значения совместной вероятности. Первый терм представляет собой условную вероятность некоторой конфигурации данных о наличии ветерка, если дана конфигурация расположения ям; он принимает значение 1, если в квадратах, соседних с ямами, чувствуется ветерок, в противном случае принимает значение 0. Вторым термом яв-

ляется априорная вероятность конфигурации расположения ям. Каждый квадрат может содержать яму с вероятностью 0,2, независимо от других квадратов, поэтому имеет место следующее:

$$\mathbf{P}(P_{1,1}, \dots, P_{4,4}) = \sum_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j}) \quad (13.15)$$

Для конфигурации с  $n$  ямами это значение равно  $0.2^n \times 0.8^{16-n}$ .

В ситуации, показанной на рис. 13.2, *a*, свидетельство состоит из наблюдаемого ветерка (или его отсутствия) в каждом посещенном квадрате, в сочетании с тем фактом, что каждый такой квадрат не содержит ямы. Эти факты можно сокращенно представить как  $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$  и  $known = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$ . Мы заинтересованы в получении ответов на такие запросы, как  $\mathbf{P}(P_{1,3} | known, b)$ : насколько велика вероятность того, что квадрат [1, 3] содержит яму, если даны результаты всех наблюдений, сделанных до сих пор?

Для получения ответа на этот запрос можно воспользоваться стандартным подходом, основанном на уравнении 13.6 и реализованном в алгоритме *Enumerate-Joint-Ask*, а именно просуммировать элементы из таблицы полного совместного распределения. Допустим, что *Unknown* (Неизвестное) — составная переменная, которая состоит из переменных  $P_{i,j}$  для квадратов, отличных от квадратов *Known* и квадрата запроса [1, 3]. В таком случае с помощью уравнения 13.6 получаем следующее:

$$\mathbf{P}(P_{1,3} | known, b) = \alpha \sum_{\text{unknown}} \mathbf{P}(P_{1,3}, \text{unknown}, known, b)$$

Полное совместное распределение вероятностей уже было задано, поэтому задача решена, точнее, осталось только выполнить вычисления. Количество неизвестных квадратов равно 12, поэтому требуемая сумма состоит из  $2^{12} = 4096$  термов. Вообще говоря, количество термов в этой сумме растет экспоненциально в зависимости от количества квадратов.

Но интуиция подсказывает, что здесь есть какое-то упущение. Безусловно, напрашивается вопрос, не являются ли другие квадраты не относящимися к делу? Ведь содержимое квадрата [4, 4] не влияет на то, имеется ли яма в квадрате [1, 3]! И действительно, эта догадка является правильной. Пользуясь ею, предположим, что *Fringe* — переменные (отличные от переменной запроса), которые описывают свойства квадратов, соседних по отношению к посещенным квадратам; в данном случае таковыми являются только [2, 2] и [3, 1]. Кроме того, допустим, что *Other* — это переменные, которые относятся к другим неизвестным квадратам; в данном случае количество других квадратов равно 10, как показано на рис. 13.2, *b*. Ключевая идея состоит в том, что данные о наблюдаемом ветерке являются условно независимыми от других переменных, если даны условные переменные, переменные периферии и переменная запроса. После этой догадки остается лишь, так сказать, дело техники — выполнить несколько алгебраических операций.

Чтобы воспользоваться этой идеей, необходимо преобразовать формулу запроса в такую форму, в которой данные о наличии ветерка становятся условно зависимыми

от всех других переменных, а затем упростить полученное выражение с использованием утверждения об условной независимости, как показано ниже.

$$\begin{aligned}
 & \mathbf{P}(P_{1,3} | \text{known}, b) \\
 &= \alpha \sum_{\text{unknown}} \mathbf{P}(b | P_{1,3}, \text{known}, \text{unknown}) \mathbf{P}(P_{1,3}, \text{known}, \text{unknown}) \quad (\text{согласно} \\
 &\qquad\qquad\qquad \text{правилу произведения}) \\
 &= \alpha \sum_{\text{fringe}} \sum_{\text{other}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe}, \text{other}) \mathbf{P}(P_{1,3}, \text{known}, \text{fringe}, \text{other}) \\
 &= \alpha \sum_{\text{fringe}} \sum_{\text{other}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe}) \mathbf{P}(P_{1,3}, \text{known}, \text{fringe}, \text{other})
 \end{aligned}$$

В этом преобразовании на конечном этапе используется утверждение об условной независимости. Теперь первый терм в выражении не зависит от других переменных, поэтому можно переместить операцию суммирования внутрь выражения, следующим образом:

$$\begin{aligned}
 & \mathbf{P}(P_{1,3} | \text{known}, b) \\
 &= \alpha \sum_{\text{fringe}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe}) \sum_{\text{other}} \mathbf{P}(P_{1,3}, \text{known}, \text{fringe}, \text{other})
 \end{aligned}$$

Согласно утверждению о независимости, соответствующему приведенному в уравнении 13.15, терм априорной вероятности может быть факторизован, после чего все эти термы могут быть переупорядочены следующим образом:

$$\begin{aligned}
 & \mathbf{P}(P_{1,3} | \text{known}, b) \\
 &= \alpha \sum_{\text{fringe}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe}) \sum_{\text{other}} \mathbf{P}(P_{1,3}) P(\text{known}) P(\text{fringe}) P(\text{other}) \\
 &= \alpha P(\text{known}) \mathbf{P}(P_{1,3}) \sum_{\text{fringe}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe}) P(\text{fringe}) \sum_{\text{other}} P(\text{other}) \\
 &= \alpha' \mathbf{P}(P_{1,3}) \sum_{\text{fringe}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe}) P(\text{fringe})
 \end{aligned}$$

В этом преобразовании на последнем этапе постоянный терм  $P(\text{known})$  вводится в нормализующую константу и используется тот факт, что выражение  $\sum_{\text{other}} P(\text{other})$  равно 1.

Теперь в сумме по периферийным переменным  $P_{2,2}$  и  $P_{3,1}$  осталось только четыре терма. Использование свойств независимости и условной независимости позволило полностью исключить из рассмотрения другие квадраты. Обратите внимание на то, что выражение  $\mathbf{P}(b | \text{known}, P_{1,3}, \text{fringe})$  равно 1, если периферия совместима с данными наблюдений о наличии ветерка, а в противном случае равно 0. Поэтому для получения каждого значения  $P_{1,3}$  проводится суммирование по логическим моделям для периферийных переменных, которые согласованы с известными фактами (это можно сравнить с тем, как осуществлялся перебор моделей на

рис. 7.4). Эти модели и связанные с ними априорные вероятности,  $P(\text{fringe})$ , показаны на рис. 13.3. Итак, получаем следующие значения:

$$\begin{aligned} P(P_{1,3} | \text{known}, b) &= \alpha' < 0.2(0.04 + 0.16 + 0.16), 0.8(0.04 + 0.16) > \approx \\ &< 0.31, 0.69 > \end{aligned}$$

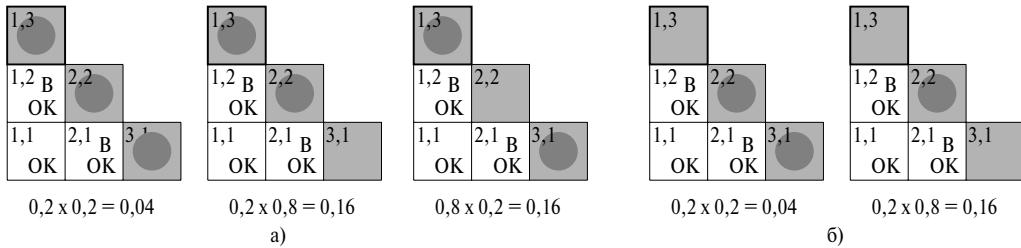


Рис. 13.3. Согласованные модели для периферийных переменных  $P_{2,2}$  и  $P_{3,1}$ , показывающие значение  $P(\text{fringe})$  для каждой модели: три модели с  $P_{1,3} = \text{true}$ , где показаны две или три ямы (а); две модели с  $P_{1,3} = \text{false}$ , где показаны одна или две ямы (б)

Это означает, что квадрат [1, 3] (а также квадрат [3, 1], поскольку для него могут быть выполнены подобные вычисления) содержит яму с вероятностью приблизительно 31%. Аналогичное вычисление, которое авторы рекомендуют выполнить читателю, показывает, что квадрат [2, 2] содержит яму с вероятностью приблизительно 86%. Агент в мире вампуса определенно должен избегать квадрата [2, 2]!

В данном разделе наглядно продемонстрировано, что даже такие задачи, которые внешне кажутся очень сложными, могут быть точно сформулированы в терминах теории вероятностей и решены с использованием простых алгоритмов. Для получения эффективных решений могут применяться соотношения, определяющие свойства независимости и условной независимости, которые позволяют упростить требуемые операции суммирования. Эти соотношения часто соответствуют нашему интуитивному пониманию того, как следует выполнять декомпозицию задачи. В следующей главе будут разработаны формальные представления для таких соотношений, а также алгоритмы, оперирующие с соответствующими представлениями и позволяющие эффективно осуществлять вероятностный вывод.

## 13.8. РЕЗЮМЕ

В данной главе было показано, что понятие вероятности лежит в основе правильных способов формирования рассуждений о неопределенности.

- Неопределенность возникает и по причине экономии усилий, и из-за отсутствия знаний. Неопределенности нельзя избежать в сложных, динамичных или труднодоступных мирах.
- Наличие неопределенности означает, что многие упрощения, возможные в дедуктивном логическом выводе, становятся больше не допустимыми.
- В оценках вероятности выражается неспособность агента прийти к определенному решению, касающемуся истинности высказывания. Вероятности являются выражением степени уверенности агента.

- К основным типам вероятностных утверждений относятся утверждения, касающиеся **априорных вероятностей и условных вероятностей** простых и сложных высказываний.
- **Полное совместное распределение вероятностей** задает вероятность каждого полного присваивания значений случайным переменным. Это распределение обычно слишком велико для того, чтобы его можно было создать или использовать в явной форме.
- Аксиомы вероятностей регламентируют возможные присваивания вероятностей высказываниям. Агент, не учитывающий в своих действиях эти аксиомы, ведет себя в некоторых обстоятельствах нерационально.
- Если полное совместное распределение доступно, оно может использоваться для получения ответов на запросы путем суммирования элементов с данными об атомарных событиях, соответствующих высказываниям запроса.
- Наличие свойства **абсолютной независимости** между подмножествами случайных переменных позволяет факторизовать полное совместное распределение на меньшие совместные распределения. Это дает возможность значительно уменьшить сложность, но редко встречается на практике.
- **Правило Байеса** позволяет вычислять неизвестные вероятности из известных условных вероятностей, обычно в причинном направлении. При наличии многочисленных свидетельств применение правила Байеса, как правило, приводит к возникновению таких же проблем масштабирования, которые возникают при использовании полного совместного распределения.
- Свойство **условной независимости**, вызванное наличием прямых причинных связей в рассматриваемой проблемной области, может обеспечить факторизацию полного совместного распределения на меньшие условные распределения. В **наивной байесовской** модели предполагается наличие условной независимости всех переменных действия, если задана одна переменная причины; размеры этой модели увеличиваются линейно, в зависимости от количества результатов.
- Агент в мире вампуша может рассчитывать вероятности ненаблюдаемых объектов мира и использовать их для принятия лучших решений по сравнению с теми, которые принимает простой логический агент.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Игры с элементами случайного выбора известны, по меньшей мере, с 300 года до н.э., но математический анализ случайных чисел и вероятностей стал проводиться гораздо позднее. Некоторые работы, выполненные индийским математиком Махавиракарье (Mahaviracarya), датированы примерно IX веком. В Европе первые попытки проведения такого анализа начались только в эпоху итальянского Возрождения, приблизительно с 1500 года. Первый значимый систематический анализ был проведен Джироламо Кардано примерно в 1565 году, но его работы оставались неопубликованными до 1663 года. В это время открытие Блезом Паскалем систематического способа вычисления вероятностей (о котором он сообщил в переписке с

Пьером Ферма в 1654 году) впервые привело к утверждению теории вероятностей как математической дисциплины. Первым опубликованным учебником по теории вероятностей была книга Гюйгенса *De Ratiociniis in Ludo Aleae* [712]. Кроме того, Паскаль ввел понятие условной вероятности, которое описано в учебнике Гюйгенса. Преподобный Томас Байес (1702–1761 гг.) сформулировал правило формирования рассуждений об условных вероятностях, которое было названо в его честь. Работы Байеса были опубликованы только после его смерти [88]. Колмогоров в своей работе [826] (которая вышла вначале на немецком языке, в 1933 году) впервые представил теорию вероятностей в виде строго аксиоматической инфраструктуры. В дальнейшем Ренъи [1282] сформулировал аксиоматическое представление, в котором в качестве примитивного понятия используется условная, а не абсолютная вероятность.

Паскаль использовал вероятность в таких вычислениях, которые требовали не только ее объективной интерпретации, как свойства мира, основанного на симметрии или относительных частотах событий, но и субъективной интерпретации, основанной на оценке степени уверенности. Первая интерпретация обнаруживается в проведенном Паскалем анализе вероятностей в играх с элементами случайности, а последняя — в знаменитых доводах “Спора с Паскалем”, касающихся возможного существования Бога. Однако Паскаль недостаточно четко учитывал различие между этими двумя интерпретациями. Указанное различие было впервые наглядно подчеркнуто Джеймсом Бернулли (1654–1705 гг.).

Лейбниц ввел “классическое” понятие вероятности как доли перечислимых, равновероятных случаев, которое использовалось также Бернулли, но полностью проанализировано Лапласом (1749–1827 гг.). Это понятие является противоречивым из-за наличия частотной и субъективной интерпретации. События могут рассматриваться как равновероятные либо из-за наличия естественной, физической симметрии между ними, либо просто из-за того, что мы не обладаем достаточными знаниями, которые позволили бы считать одно событие более вероятным, чем другое. Подход, предусматривающий использование последних, субъективных соображений, оправдывающих допустимость присваивания равных вероятностей, известен под названием *принципа безразличия* [792].

Споры между приверженцами объективного и субъективного подходов обострились в XX столетии. Колмогоров [827], Р.А. Фишер [473] и Ричард фон Мизес [1544] были сторонниками относительной частотной интерпретации. Приведенная в работе Карла Поппера [1228] (которая была вначале выпущена на немецком языке, в 1934 году) интерпретация “проявлений закономерностей” позволяет проследить истоки формирования относительных частот вплоть до основополагающих законов физической симметрии. Франк Рамсей [1265], Бруно де Финетти [342], Р.Т. Кокс [304], Леонард Сэвеж [1354] и Ричард Джеффри [727] интерпретировали вероятности как степени уверенности конкретных лиц. Проводимый ими анализ степеней уверенности был тесно связан с полезностями и с поведением, а именно в этом направлении анализа оценивалась готовность субъекта делать те или иные ставки. Рудольф Карнап, последовав за Лейбницем и Лапласом, предложил субъективную интерпретацию вероятности другого рода — не как определенной степени уверенности конкретного лица, а как степень уверенности, которую должна иметь идеализированная личность в отношении истинности конкретного высказывания *a*, если дан конкретный ряд свидетельств *e*. Карнап попытался продвинуться дальше Лейбница или Лапласа, сформулировав понятие степени  подтверждения математиче-

ски точно, как логического отношения между *a* и *e*. Исследования этого отношения имели своей целью создание математической дисциплины, которая была названа **индуктивной логикой** по аналогии с обычной дедуктивной логикой [224], [225]. Карнап не смог далеко расширить свою индуктивную логику за пределы пропозиционального случая, а Хилари Патнем [1247] показала, что некоторые фундаментальные сложности послужили бы препятствием при создании строгих расширений языков, способных выражать арифметические соотношения.

С попыткой создать индуктивную логику тесно связана проблема референтных классов. Подход, предусматривающий выбор “наиболее конкретного” референтного класса с достаточными размерами, был формально предложен Рейхенбахом [1273]. Были сделаны различные попытки сформулировать более сложные подходы для предотвращения некоторых очевидных ошибок, связанных с использованием правила Рейхенбаха (к наиболее известным из них относятся работы Генри Кайберга [873], [874]), но такие подходы остаются в определенной степени лишенными теоретического фундамента. В опубликованной в дальнейшем работе [54] методы Карнапа распространены на теорию первого порядка, что позволило избежать многих сложностей, связанных с применением прямолинейного метода референтных классов.

Байесовские вероятностные рассуждения использовались в искусственном интеллекте начиная с 1960-х годов, особенно в области медицинской диагностики. Эти методы применялись не только для составления диагноза на основании доступных свидетельств, но и для выбора дополнительных вопросов и тестов с использованием теории информационного значения (раздел 16.6), если доступные свидетельства не позволяли прийти к окончательному выводу [583], [584]. Одна из систем превзошла людей-экспертов в области диагностики острых брюшных заболеваний [340]. Но такие ранние байесовские системы были подвержены многочисленным недостаткам. Поскольку в этих системах отсутствовали какие-либо теоретические модели диагностируемых ими условий, они были чувствительными к нерепрезентативным данным, встречающимся в тех ситуациях, когда доступны были лишь небольшие выборки [341]. Еще более важным недостатком было то, что в этих системах не применялись лаконичные формальные средства (подобные тем, которые будут описаны в главе 14) для представления и использования информации об условной независимости, поэтому эксплуатация этих систем требовала накопления, хранения и обработки таблиц с вероятностными данными, достигающих колоссальных размеров. Из-за этих сложностей вероятностные методы решения задач в условиях неопределенности не вызывали интереса исследователей в области искусственного интеллекта в период 1970-х–середины 1980-х годов. Достижения в этой области, появившиеся в конце 1980-х годов, которые изменили эту ситуацию, описаны в следующей главе.

Наивное байесовское представление для совместных распределений широко исследовалось в литературе по распознаванию образов с 1950-х годов [421]. Кроме того, такой способ представления использовался в области выборки текста, часто не совсем продуманно, начиная с [984]. Вероятностные основы этого метода, который дополнительно описан в упр. 13.18, были сформулированы в работе Робертсона и Спарка Джонса [1297]. Домингос и Паццани [402] объяснили причины поразительного успеха наивных байесовских рассуждений даже в тех проблемных областях, где они явно нарушили предположения о независимости.

По теории вероятностей есть много хороших вводных учебников, включая [254] и [1310]. Морис де Грот [378] подготовил объединенное введение в проблематику ве-

роятностей и статистики с байесовской точки зрения, а также выпустил более расширенный учебник [377]. В учебнике Ричарда Хемминга [608] дано математически сложное введение в теорию вероятностей с точки зрения интерпретации проявления закономерностей, основанной на физической симметрии. В [605] и [606] описана ранняя история развития понятия вероятности. Бернштейн [114] составил увлекательный популярный отчет об истории понятия риска.

## УПРАЖНЕНИЯ

---

- 13.1. Докажите на основании основных принципов, что  $P(a | b \wedge a) = 1$ .
- 13.2. С использованием аксиом вероятностей докажите, что любое распределение вероятностей дискретной случайной переменной должно в сумме составлять 1.
- 13.3. Было бы рациональным для агента придерживаться трех убеждений —  $P(A) = 0.4$ ,  $P(B) = 0.3$  и  $P(A \vee B) = 0.5$ ? Если это так, то какой диапазон вероятностей был бы рациональным для агента применительно к  $A \wedge B$ ? Составьте таблицу, подобную приведенной в табл. 13.1, и покажите, подтверждает ли она ваши доводы в отношении рациональности. Затем составьте еще одну версию этой таблицы, в которой  $P(A \vee B) = 0.7$ . Объясните, почему рационально иметь уверенность в наличии этой вероятности, даже несмотря на то, что таблица показывает один случай, который соответствует проигрышу, и три случая, в которых достигается ничья. (*Подсказка.* Какие обязательства взял на себя агент 1 применительно к вероятности каждого из четырех случаев, в частности, того случая, когда имеет место проигрыш?)
- 13.4. Этот вопрос касается свойств атомарных событий, как описано на с. 629.
  - a) Докажите, что дизъюнкция всех возможных атомарных событий логически эквивалентна значению *true*. (*Подсказка.* Используйте доказательство по индукции, которое распространяется на произвольное количество случайных переменных.)
  - b) Докажите, что любое высказывание логически эквивалентно дизъюнкции атомарных событий, из которых следует истинность этого высказывания.
- 13.5. Рассмотрите проблемную область, в которой речь идет о раздаче в покере на руки по 5 карт из стандартной колоды в 52 карты, на основе предположения, что раздача проводится честно.
  - a) Каково количество атомарных событий в совместном распределении вероятностей (т.е. каково количество различных раздач по 5 карт)?
  - b) Какова вероятность каждого атомарного события?
  - c) Какова вероятность получения королевского флеша стрит? Четырех королей?
- 13.6. Исходя из полного совместного распределения, приведенного в табл. 13.2, вычислите следующее:
  - a)  $P(\text{toothache})$
  - b)  $P(\text{Cavity})$

- в)  $P(Toothache | cavity)$   
 г)  $P(Cavity | toothache \vee catch)$
- 13.7. Покажите, что три формы описания свойства независимости, приведенные в уравнении 13.8, являются эквивалентными.
- 13.8. После ежегодного медицинского осмотра некоторого пациента у врача есть плохая новость и хорошая новость. Плохая новость состоит в том, что оказалась положительной проверка на наличие серьезного заболевания и что точность результатов проверки составляет 99% (это означает, что вероятность получения положительного результата проверки, если пациент имеет это заболевание, равна 0,99, и такова же вероятность получения отрицательных результатов проверки, если пациент не имеет этого заболевания). Хорошая новость состоит в том, что это заболевание — редкое и поражает только 1 из 10 000 людей того возраста, в котором находится пациент. Почему новость, что это заболевание редкое, названа хорошей? Каковы шансы на то, что пациент действительно имеет данное заболевание?
- 13.9. Очень часто бывает полезно рассмотреть результаты некоторых конкретных высказываний в контексте определенного общего фонового свидетельства, которое остается неизменным, а не действовать в условиях полного отсутствия информации. В приведенных ниже вопросах предлагается доказать более общие версии правила произведения и правила Байеса применительно к некоторому фоновому свидетельству  $e$ .
- Докажите версию общего правила произведения с условными вероятностями:  

$$P(X, Y | e) = P(X | Y, e) \cdot P(Y | e)$$
  - Докажите версию правила Байеса из уравнения 13.10 с условными вероятностями.
- 13.10. Покажите, что утверждение  

$$P(A, B | C) = P(A | C) \cdot P(B | C)$$
- эквивалентно любому из следующих утверждений:
- $$P(A | B, C) = P(A | C) \quad \text{и} \quad P(B | A, C) = P(B | C)$$
- 13.11. Допустим, что вам вручили мешок, содержащий  $n$  подлинных монет, и сообщили, что  $n-1$  из этих монет являются нормальными, такими, что с одной стороны имеется орел, с другой — решка, а одна монета — фальшивая, и с обеих ее сторон имеется орел.
- Предположим, что вы открыли мешок, случайным образом выбрали монету, не пытаясь определить ее свойства, подбросили ее, после чего выпал орел. Какова (условная) вероятность того, что выбранная вами монета является фальшивой?
  - Предположим, что вы продолжаете подбрасывать монету всего  $k$  раз после того, как ее выбрали, и обнаруживаете  $k$  выпадений орла. Какова теперь условная вероятность того, что вы выбрали фальшивую монету?
  - Предположим, что вы хотите принять решение в отношении того, является ли выбранная монета фальшивой, подбросив ее  $k$  раз. Процедура

принятия решения возвращает FAKE (Фальшивая), если все  $k$  бросков приводят к выпадению орла, а в противном случае возвращает NORMAL (Нормальная). Какова (безусловная) вероятность того, что эта процедура допускает ошибку?

- 13.12.** В этом упражнении предлагается выполнить вычисление коэффициента нормализации для примера с менингитом. Вначале выберите подходящее значение для  $P(S|-\bar{M})$  и примените его для вычисления ненормализованных значений  $P(M|S)$  и  $P(\neg M|S)$  (т.е. игнорируя терм  $P(S)$  в выражении правила Байеса). Теперь нормализуйте эти значения таким образом, чтобы они в сумме составляли 1.
- 13.13.** В этом упражнении исследуется то, как влияют соотношения, касающиеся условной независимости на количество информации, требуемой для вероятностных вычислений.
- Предположим, что необходимо рассчитать значение  $P(h|e_1, e_2)$ , а информация об условной независимости отсутствует. Какие из следующих множеств чисел являются достаточными для такого вычисления?
    - $P(E_1, E_2)$ ,  $P(H)$ ,  $P(E_1|H)$ ,  $P(E_2|H)$
    - $P(E_1, E_2)$ ,  $P(H)$ ,  $P(E_1, E_2|H)$
    - $P(H)$ ,  $P(E_1|H)$ ,  $P(E_2|H)$
  - Предположим, известно, что  $P(E_1|H, E_2) = P(E_1|H)$  для всех значений  $H$ ,  $E_1$ ,  $E_2$ . Какое из этих трех множеств значений теперь будет достаточным?
- 13.14.** Допустим, что  $X$ ,  $Y$ ,  $Z$  — булевые случайные переменные. Обозначьте восемь элементов в совместном распределении  $P(X, Y, Z)$  буквами алфавита от  $a$  до  $h$ . Выразите утверждение, что  $X$  и  $Y$  являются условно независимыми, если дано  $Z$ , в виде множества уравнений, связывающих элементы от  $a$  до  $h$ . Сколько среди них неизбыточных уравнений?
- 13.15.** (*Адаптировано из [1191].*) Предположим, что вы — свидетель ночного наезда на пешехода в Афинах с участием такси, виновник которого уехал с места происшествия. Все такси в Афинах покрашены в синий или зеленый цвет. Вы поклялись под присягой, что такси было синим, а результаты широких экспериментов показывают, что в условиях плохого освещения надежность распознавания синего и зеленого цветов составляет 75%. Возможно ли рассчитать наиболее вероятный цвет этого такси? (*Подсказка.* Тщательно проведите различие между высказыванием, что такси — синего цвета, и высказыванием, что оно внешне казалось синим.)
- А что можно сказать, получив информацию о том, что в Афинах 9 из 10 такси имеют зеленый цвет?
- 13.16.** (*Адаптировано из [1191].*) Три заключенных,  $A$ ,  $B$  и  $C$ , заперты в своих камерах. Всем известно, что один из них завтра будет казнен, а другие помилованы. Но только губернатор знает, кто именно будет казнен. Заключенный  $A$  просит охранника об одолжении: “Пожалуйста, узнайте у губернатора, кто будет казнен, а затем передайте сообщение одному из моих друзей,  $B$  или  $C$ , чтобы он знал, что утром будет помилован”. Охранник соглашается, а после возвращения говорит заключенному  $A$ , что передал сообщение о помиловании заключенному  $B$ .

Каковы шансы заключенного А на то, что он будет казнен, при наличии этой информации? (Ответьте на этот вопрос с помощью математики, а не энергичной жестикуляции.)

- 13.17.** Напишите общий алгоритм получения ответов на запросы в форме  $\mathbf{P}(\text{Cause}|\mathbf{e})$  с использованием наивного байесовского распределения. Вы должны исходить из предположения, что свидетельство  $\mathbf{e}$  может присваивать значения любому подмножеству переменных результата.
- 13.18.** Категоризацией текста называется задача присваивания данному конкретному документу одной из фиксированного множества категорий на основе анализа содержащегося в нем текста. Для решения этой задачи часто используются наивные байесовские модели. В этих моделях переменной запроса является категория документа, а в качестве переменных “результата” рассматривается наличие или отсутствие каждого слова из языка документа; основное предположение состоит в том, что слова в документах встречаются независимо друг от друга, а их частоты определяются категорией документа.
- a)** Объясните точно, как можно сформировать такую модель, получив в качестве “обучающих данных” множество документов, которые были распределены по категориям.
  - б)** Объясните точно, как следует определять категорию нового документа.
  - в)** Является ли указанное предположение о независимости обоснованным? Обсудите этот вопрос.
- 13.19.** В проведенном в данной главе анализе мира вампуса использовался тот факт, что каждый квадрат содержит яму с вероятностью 0,2, независимо от содержимого других квадратов. Вместо этого примите предположение, что точно  $N/5$  ям разбросаны равномерно случайным образом среди  $N$  квадратов, отличных от  $[1, 1]$ . Остаются ли переменные  $P_{i,j}$  и  $P_{k,1}$  все еще независимыми? Каково теперь совместное распределение  $\mathbf{P}(P_{1,1}, \dots, P_{4,4})$ ? Снова проведите вычисления вероятностей наличия ям в квадратах  $[1, 3]$  и  $[2, 2]$ .

# 14 ВЕРОЯТНОСТНЫЕ РАССУЖДЕНИЯ

*В этой главе описано, как составлять сетевые модели для формирования рассуждений в условиях неопределенности в соответствии с законами теории вероятностей.*

В главе 13 рассматривались синтаксис и семантика теории вероятностей. В ней отмечалась важность связей, определяющих независимость и условную независимость высказываний, для упрощения вероятностных представлений о мире. В данной главе описан систематический способ явного представления таких связей в форме **байесовских сетей**. В ней будут определены синтаксис и семантика этих сетей и показано, как они могут использоваться для представления неопределенных знаний естественным и эффективным способом. Затем в этой главе будет продемонстрировано, что вероятностный вывод, хотя и не осуществимый с помощью вычислительных методов в наихудшем случае, может эффективно выполняться во многих практических ситуациях. Кроме того, будет описан целый ряд алгоритмов приближенного вероятностного вывода, которые часто могут применяться в тех случаях, когда точный вероятностный вывод неосуществим. В этой главе будут также представлены способы, с помощью которых теория вероятностей может быть применена к мирам с объектами и отношениями, т.е. к представлениям первого порядка, в отличие от пропозициональных представлений. Наконец, в данной главе приведен обзор альтернативных подходов к формированию рассуждений в условиях неопределенности.

## 14.1. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В НЕОПРЕДЕЛЕННОЙ ПРОБЛЕМНОЙ ОБЛАСТИ

В главе 13 было показано, что полное совместное распределение вероятностей позволяет отвечать на любые вопросы о рассматриваемой проблемной области, но может приобретать по мере увеличения количества переменных настолько большие размеры, что вычисления становятся невозможными. Более того, сам способ задания вероятностей для атомарных событий является довольно неестественным и

может оказаться весьма затруднительным при отсутствии большого объема данных, на основании которых накапливаются статистические оценки.

Кроме того, в предыдущей главе было показано, что связи, определяющие независимость и условную независимость между переменными, позволяют намного сократить количество вероятностей, которые должны быть заданы в целях определения полного совместного распределения. В настоящем разделе показана структура данных, называемая<sup>1</sup> **байесовской сетью**, которая позволяет представить связи между переменными и составить краткую спецификацию любого полного совместного распределения вероятностей.

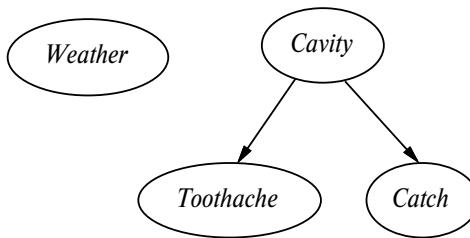
Байесовская сеть — это ориентированный граф, в котором каждая вершина помечена количественной вероятностной информацией. Полная спецификация такой сети описана ниже.

1. Вершинами сети является множество случайных переменных. Переменные могут быть дискретными или непрерывными.
2. Вершины соединяются попарно ориентированными ребрами, или ребрами со стрелками; ребра образуют множество ребер. Если стрелка направлена от вершины  $X$  к вершине  $Y$ , то вершина  $X$  называется родительской вершиной вершины  $Y$ .
3. Каждая вершина  $X_i$  характеризуется распределением условных вероятностей  $P(X_i | \text{Parents}(X_i))$ , которое количественно оценивает влияние родительских вершин на эту вершину.
4. Граф не имеет циклов, состоящих из ориентированных ребер (и поэтому является ориентированным ациклическим графом (Directed Acyclic Graph — DAG)).

Топология сети (множество вершин и ребер) показывает отношения, определяющие условную независимость, которые проявляются в данной проблемной области, в том смысле, который вскоре будет точно сформулирован. Интуитивный смысл стрелки в правильно составленной сети обычно состоит в том, что вершина  $X$  оказывает непосредственное влияние на вершину  $Y$ . Для специалиста в проблемной области задача определения того, какие непосредственные влияния существуют в этой проблемной области, обычно является довольно легкой; действительно, она намного легче по сравнению с фактическим определением самих вероятностей. После того как составлена топология байесовской сети, остается только указать распределение условных вероятностей для каждой переменной с учетом ее родительских переменных. В данной главе будет показано, что применение этой топологии и распределений условных вероятностей вполне позволяет (неявно) задать полное совместное распределение для всех переменных.

<sup>1</sup> Такое название применяется наиболее часто, но для обозначения этой структуры данных предусмотрено также много других названий, в том числе **сеть доверия** (belief network), **вероятностная сеть** (probabilistic network), **причинная сеть** (causal network) и **схема знаний** (knowledge map). В статистике термином **графическая модель** (graphical model) обозначается немного более широкий класс структур данных, который включает байесовские сети. Еще одно расширение байесовских сетей, называемое **сетью принятия решений** (decision network), или **диаграммой влияния** (influence diagram), рассматривается в главе 16.

Еще раз вернемся к простому миру, описанному в главе 13, который состоит из переменных *Toothache*, *Cavity*, *Catch* и *Weather*. В этой главе было показано, что переменная *Weather* не зависит от других переменных; более того, было продемонстрировано, что переменные *Toothache* и *Catch* являются условно независимыми, если задана переменная *Cavity*. Эти отношения представлены в виде структуры байесовской сети, показанной на рис. 14.1. Формально условная независимость переменных *Toothache* и *Catch*, если задана переменная *Cavity*, обозначается отсутствием связи между *Toothache* и *Catch*. Интуитивно можно понять, что в сети представлен факт — *Cavity* является непосредственной причиной *Toothache* и *Catch*, тогда как между *Toothache* и *Catch* не существует прямой причинной связи.



*Рис. 14.1. Простая байесовская сеть, в которой переменная Weather независима от трех других переменных, а переменные Toothache и Catch являются условно независимыми, если задана переменная Cavity*

Теперь рассмотрим следующий пример, который является немного более сложным. Житель пригорода установил в своем доме новую систему тревожной сигнализации для обнаружения взлома. Она довольно надежно обнаруживает взлом, но иногда также реагирует на небольшие землетрясения. (Этим примером мы обязаны Джуди Перлу, который живет в Лос-Анджелесе, поэтому проявляет острый интерес к землетрясениям.) У этого человека есть два соседа, Джон и Мэри, которые обещали звонить ему на работу, услышав тревожный сигнал. Джон всегда звонит, услышав тревожный сигнал, но иногда путает с ним телефонный звонок в доме соседа и в этих случаях также звонит. Мэри любит слушать довольно громкую музыку и поэтому иногда вообще пропускает тревожный сигнал. Получив факты о том, кто из этих соседей звонил или не звонил, необходимо оценить вероятность взлома. Байесовская сеть для этой проблемной области приведена на рис. 14.2.

На время отвлечемся от распределения условных вероятностей, показанных на этом рисунке, и сосредоточимся на топологии сети. В случае сети определения взлома топология показывает, что взлом и землетрясения непосредственно влияют на вероятность появления тревожного сигнала, а звонки Джона и Мэри зависят только от тревожного сигнала. Поэтому сеть подтверждает наши предположения, что соседи самостоятельно не обнаруживают какие-либо попытки взлома, не замечают незначительных землетрясений и не совещаются друг с другом перед звонками.

Обратите внимание на то, что в этой сети нет вершин, соответствующих тем ситуациям, в которых Мэри в настоящее время слушала бы громкую музыку или звонил бы телефон и сбивал с толку Джона. Эти факторы подытожены в показателях

неопределенности, связанных с ребрами, направленными от вершины *Alarm* к вершинам *JohnCalls* и *MaryCalls*. Такая структура сети служит примером проявления в действии не только экономии усилий, но и недостатка знаний, поскольку потребовалось бы слишком много работы, чтобы узнать, по какой причине эти факторы могут оказаться более или менее вероятными в каждом конкретном случае; к тому же все равно отсутствует приемлемый способ получения релевантной информации. Вероятности, показанные на рисунке, фактически подытоживают потенциально бесконечное множество обстоятельств, которые либо могут вызвать нарушения при выработке тревожного сигнала (высокая влажность, отказ сети электропитания, разрядка аккумулятора, обрыв проводов, дохлая мышь, застрявшая внутри звонка, и т.д.), либо станут причиной того, что Джон или Мэри не смогут о нем сообщить (из-за того, что выйдут на обед, отправятся в отпуск, на время оглохнут, не рассышат сигнал в шуме пролетающего вертолета и т.д.). Но именно благодаря использованию приближенных оценок маленький агент получает возможность узнавать, что происходит в большом мире, по крайней мере, приблизительно. Степень приближения к истине может быть повышена по мере введения дополнительной релевантной информации.

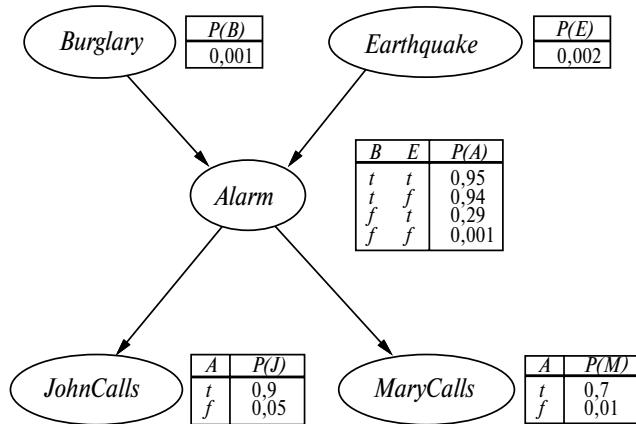


Рис. 14.2. Типичная байесовская сеть, на которой показаны и топология, и таблицы условных вероятностей (Conditional Probability Table — CPT). В таблицах CPT буквами В, Е, А, І и М обозначены следующие события: Burglary (Взлом), Earthquake (Землетрясение), Alarm (Тревожный сигнал), JohnCalls (Звонки Джона) и MaryCalls (Звонки Мэри)

Теперь обратимся к распределениям условных вероятностей, показанным на рис. 14.2. На этом рисунке каждое распределение представлено в виде **таблицы условных вероятностей**, или сокращенно CPT (Conditional Probability Table). (Такая форма таблицы может использоваться для дискретных переменных; другие представления, включая те, которые подходят для непрерывных переменных, описаны в разделе 14.2.) Каждая строка в таблице CPT содержит условную вероятность каждого значения вершины для **обусловливающего случая** (conditioning case), определяющего условную вероятность. Обусловливающий случай представляет собой одну из возможных комбинаций значений родительских вершин (в принципе его можно

рассматривать как миниатюрное атомарное событие). Каждая строка должна в сумме составлять 1, поскольку элементы этой строки представляют собой исчерпывающее множество случаев для данной переменной. А если речь идет о булевых переменных, то после определения вероятности истинного значения, скажем  $p$ , вероятность ложного значения должна быть равна  $1-p$ , поэтому в таблицах СРТ второе число часто не указывают, как и на рис. 14.2. Вообще говоря, любая таблица для булевой переменной с  $k$  булевыми родительскими переменными содержит  $2^k$  независимо определяемых вероятностей. Таблица для вершины без родительских вершин имеет только одну строку, представляющую априорные вероятности каждого возможного значения соответствующей переменной.

## 14.2. СЕМАНТИКА БАЙЕСОВСКИХ СЕТЕЙ

В предыдущем разделе описано, какой должна быть байесовская сеть, но не указано, что означает эта сеть. Существуют два способа, позволяющих понять семантику байесовских сетей. Первый из них состоит в том, что сеть следует считать одним из представлений совместного распределения вероятностей, а второй — в том, что она должна рассматриваться как описание совокупности утверждений об условной независимости. Эти две трактовки являются эквивалентными, но первая оказалась более полезной при определении способа составления сетей, а вторая — более применимой при проектировании процедур вероятностного вывода.

### Представление полного совместного распределения

Любая байесовская сеть представляет собой полное описание рассматриваемой проблемной области. Каждый элемент в полном совместном распределении вероятностей (которое ниже будет сокращенно именоваться “совместным распределением”) может быть рассчитан на основании информации, представленной в этой сети. Универсальным элементом в совместном распределении является вероятность конъюнкции конкретных присваиваний значений каждой переменной, такой как  $P(X_1=x_1 \wedge \dots \wedge X_n=x_n)$ . В качестве сокращенного обозначения для такой конъюнкции будет использоваться выражение  $P(x_1, \dots, x_n)$ . Значение этого элемента задается следующей формулой:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | parents(X_i)) \quad (14.1)$$

где  $parents(X_i)$  обозначает конкретные значения переменных в множестве вершин  $Parents(X_i)$ . Поэтому каждый элемент в совместном распределении представлен в виде произведения соответствующих элементов в таблицах условных вероятностей (Conditional Probability Table — СРТ) байесовской сети. Таким образом, таблицы СРТ обеспечивают декомпонованное представление совместного распределения.

Для иллюстрации описанных выше понятий рассчитаем вероятность того, что прозвучал тревожный сигнал, но не произошли ни взлом, ни землетрясение, а позвонили и Мэри, и Джон. Мы будем использовать однобуквенные имена для этих переменных (см. рис. 14.2):

$$\begin{aligned}
 P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) \\
 &= P(j|a) P(m|a) P(a|\neg b \wedge \neg e) P(\neg b) P(\neg e) \\
 &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062
 \end{aligned}$$

В разделе 13.4 было описано, как можно использовать полное совместное распределение для получения ответа на любой запрос о данной проблемной области. А если байесовская сеть служит представлением совместного распределения, то может также применяться для получения ответа на любой запрос путем суммирования соответствующих элементов совместного распределения. В разделе 14.4 показано, как найти ответ с помощью такого способа, а также описаны методы, которые являются гораздо более эффективными.

### Метод составления байесовских сетей

В уравнении 14.1 показано, что означает данная конкретная байесовская сеть. Но это уравнение не позволяет определить, как следует составлять байесовскую сеть таким образом, чтобы результирующее совместное распределение служило адекватным представлением данной проблемной области. Тем не менее в этом разделе будет показано, что из уравнения 14.1 следуют определенные отношения условной независимости, которые могут использоваться инженером по знаниям в качестве руководящих указаний при составлении топологии сети. Вначале перезапишем это совместное распределение в терминах условных вероятностей с использование правила произведения (см. главу 13):

$$P(x_1, \dots, x_n) = P(x_n|x_{n-1}, \dots, x_1) P(x_{n-1}, \dots, x_1)$$

Затем повторим этот процесс, приводя каждую конъюнктивную вероятность к условной вероятности и меньшей конъюнкции. В конечном итоге будет получено одно большое произведение:

$$\begin{aligned}
 P(x_1, \dots, x_n) &= P(x_n|x_{n-1}, \dots, x_1) P(x_{n-1}|x_{n-2}, \dots, x_1) \dots P(x_2|x_1) P(x_1) \\
 &= \prod_{i=1}^n P(x_i | x_{i-1}, \dots, x_1)
 \end{aligned}$$

Это тождество справедливо для любого множества случайных переменных и называется **цепным правилом** (chain rule). Сравнивая его с уравнением 14.1, можно обнаружить, что данная спецификация совместного распределения эквивалентна общему утверждению, что для каждой переменной  $X_i$  в сети верно следующее:

$$\mathbf{P}(X_i | X_{i-1}, \dots, X_1) = \mathbf{P}(X_i | \text{Parents}(X_i)) \quad (14.2)$$

при условии, что  $\text{Parents}(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$ . Это последнее условие можно выполнить, разметив вершины графа в любом порядке, совместимом с частичным упорядочением, неявно заданным в структуре графа.

Фактически уравнение 14.2 свидетельствует о том, что байесовская сеть служит правильным представлением проблемной области, только если каждая вершина в ней условно независима от ее предшественников в конкретном упорядочении вершин, после того как заданы ее родительские вершины. Поэтому, для того чтобы составить байесовскую сеть с правильной структурой для рассматриваемой проблемной области, необходимо выбрать для каждой вершины родительские вершины так, чтобы соблюдалось это свойство. Интуитивно ясно, что множество родительских

вершин вершины  $X_i$  должно включать все такие вершины из множества  $X_1, \dots, X_{i-1}$ , которые ~~непосредственно влияют~~ на  $X_i$ . Например, предположим, что мы полностью составили сеть, показанную на рис. 14.2, и осталось только выбрать родительские вершины для *MaryCalls*. Безусловно, на вершину *MaryCalls* оказывает влияние то, произошло ли событие *Burglary* или *Earthquake*, но это — не непосредственное влияние. Очевидно, что наши знания в этой проблемной области говорят о том, что эти события влияют на поведение Мэри, касающееся звонков, только через свое воздействие на тревожный сигнал. Кроме того, если речь идет о наличии тревожного сигнала, то звонок Джона не влияет на звонок Мэри. Формально говоря, составляя эту сеть, мы уверены в том, что справедливо следующее утверждение об условной независимости:

$$\begin{aligned} \mathbf{P}(\text{MaryCalls} | \text{JohnCalls}, \text{Alarm}, \text{Earthquake}, \text{Burglary}) &= \\ \mathbf{P}(\text{MaryCalls} | \text{Alarm}) \end{aligned}$$

### Компактность сети и упорядочение вершин

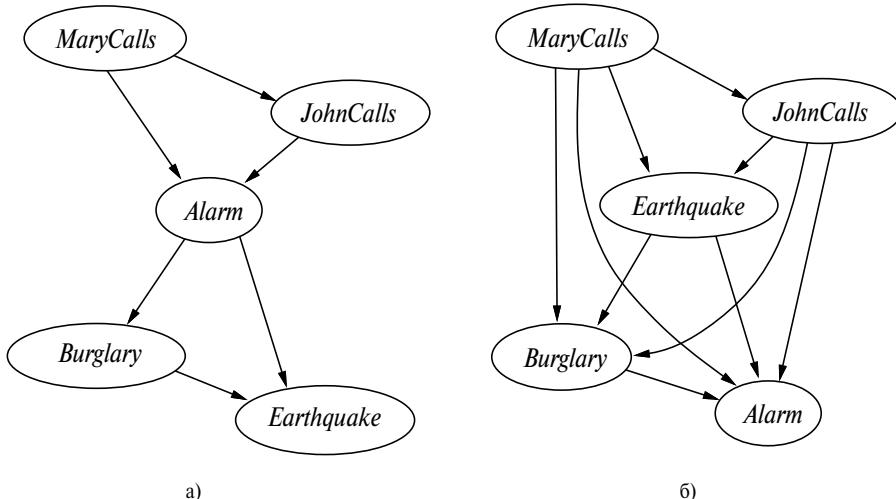
Байесовская сеть не только является полным и неизбыточным представлением проблемной области, но часто оказывается намного более компактной по сравнению с полным совместным распределением. Именно благодаря этому свойству байесовские сети становятся применимыми для представления проблемных областей со многими переменными. Компактность байесовских сетей является примером очень общего свойства ~~локально структурированных~~ систем (называемых также ~~разреженными~~ системами). В локально структурированной системе каждый субкомпонент непосредственно взаимодействует только с ограниченным количеством других компонентов независимо от общего количества компонентов. Локальная структура обычно ассоциируется с линейным, а не с экспоненциальным ростом сложности. В случае байесовских сетей резонно предположить, что в большинстве проблемных областей на каждую случайную переменную оказывают непосредственное влияние самое большое  $k$  других переменных, где  $k$  — некоторая константа. Если для простоты предположить, что в сети представлено  $n$  булевых переменных, то количество информации, необходимое для задания каждой таблицы условных вероятностей, будет составлять не больше  $2^k$  чисел, а полная сеть может быть определена с помощью  $n2^k$  чисел. В отличие от этого, совместное распределение содержит  $2^n$  чисел. В качестве конкретного примера предположим, что имеется  $n=30$  вершин, а каждая из них имеет пять родительских вершин ( $k=5$ ). В таком случае для соответствующей байесовской сети потребуется 960 чисел, а для полного совместного распределения — больше миллиарда.

Существуют и такие проблемные области, в которых на каждую переменную могут оказывать непосредственное влияние все другие переменные, поэтому соответствующая сеть является полностью связной. В таком случае для задания таблиц условных вероятностей требуется такое же количество информации, как и для задания совместного распределения. Но есть и такие проблемные области, в которых существуют незначительные зависимости, тем не менее, обязательно требующие включения в сеть путем добавления нового ребра. Но если эти зависимости выражены очень слабо, то может не иметь смысла дополнительно повышать сложность сети ради небольшого выигрыша в точности. Например, можно было бы раскритиковать структуру нашей сети определения взлома на том основании, что если бы было землетря-

сение, то Джон и Мэри не позвонили бы, даже услышав тревожный сигнал, поскольку посчитали бы, что его причиной является землетрясение. Решение вопроса о том, следует ли вводить связи от *Earthquake* к *JohnCalls* и *MaryCalls* (и таким образом увеличивать размеры таблиц), зависит от результатов сравнения важности получения более точных вероятностей с затратами на указание дополнительной информации.

Составление локально структурированной байесовской сети даже в локально структурированной проблемной области представляет собой нетривиальную задачу. Для этого требуется не только знать, что на каждую переменную непосредственно влияют лишь несколько других переменных, но и убедиться в том, что топология сети действительно отражает эти непосредственные влияния, представленные с помощью соответствующего множества родительских вершин. В связи с тем, как организована сама процедура составления сети, “непосредственно влияющие” вершины должны быть введены в сеть первыми, если они затем станут родительскими вершинами тех вершин, на которые влияют. Поэтому *правильный порядок, в котором следует вводить вершины, состоит в том, что вначале необходимо вводить вершины “коренных причин”, затем вершины переменных, на которые они влияют*, и т.д. до тех пор, пока не будут достигнуты “листовые вершины”, которые не оказывают непосредственного причинного влияния на другие переменные.

А что произойдет, если будет выбран порядок, который окажется неправильным? Еще раз рассмотрим пример со взломом. Предположим, что мы решили вводить вершины в порядке *MaryCalls*, *JohnCalls*, *Alarm*, *Burglary*, *Earthquake*. В таком случае будет получена немного более сложная сеть, показанная на рис. 14.3, а). При этом процесс введения вершин происходит, как описано ниже.



*Рис. 14.3. Пример того, что структура сети зависит от порядка введения вершин. В каждой из сетей, показанных на этом рисунке, вершины вводились в порядке сверху вниз*

- Введение вершины *MaryCalls* — родительские вершины отсутствуют.
- Введение вершины *JohnCalls* — если звонит Мэри, это, по-видимому, означает, что раздался тревожный сигнал, а вероятность такого события, оче-

видно, будет выше, если позвонит также и Джон. Поэтому для вершины *JohnCalls* необходимо использовать в качестве родительской вершину *MaryCalls*.

- Введение вершины *Alarm* — безусловно, если позвонили оба соседа, вероятность того, что раздался тревожный сигнал, больше, чем лишь при одном звонке или вообще без звонков, поэтому в качестве родительских необходимо включить обе вершины, и *MaryCalls*, и *JohnCalls*.
- Введение вершины *Burglary* — если известно состояние тревожного сигнала, то звонок от Джона или Мэри может дать жильцу охраняемого дома информацию о том, что звонил его телефон или Мэри слушала музыку, но не о взломе:

$$\mathbf{P}(\text{Burglary}|\text{Alarm}, \text{JohnCalls}, \text{MaryCalls}) = \mathbf{P}(\text{Burglary}|\text{Alarm})$$

Поэтому для использования в качестве родительской вершины требуется только *Alarm*.

- Введение вершины *Earthquake* — если раздался тревожный сигнал, то возможно также, что было землетрясение (эта тревожная сигнализация является к тому же своего рода детектором землетрясения). Но известно, что если был взлом, то тревожный сигнал объясняется именно этим, а вероятность землетрясения должна быть лишь ненамного выше нормальной. Поэтому в качестве родительских вершин для этой вершины необходимо иметь и *Alarm*, и *Burglary*.

Результирующая сеть имеет на два ребра больше по сравнению с первоначальной сетью, показанной на рис. 14.2, а также требует указания трех дополнительных вероятностей. Но что еще хуже, некоторые из ее связей представляют надуманные отношения, которые требуют формирования сложных и неестественных суждений о вероятностях, таких как оценки вероятности *Earthquake*, если даны *Burglary* и *Alarm*. Такой феномен является весьма распространенным и связан с различием между причинными и диагностическими моделями, которые были представлены в главе 8. Если мы попытаемся построить диагностическую модель со связями от симптомов к причинам (как, например, от *MaryCalls* к *Alarm* или от *Alarm* к *Burglary*), то в конечном итоге будем вынуждены задавать дополнительные зависимости между причинами, которые во всем остальном являются независимыми (а часто даже между отдельно возникающими симптомами).  *Если же мы будем придерживаться причинной модели, то в конечном итоге нам придется задавать меньшие чисел и сами эти числа, скорее всего, можно будет легче определить.* Например, Тверский и Канеман [1523] для проблемной области медицинской диагностики показали, что опытные врачи предпочитают составлять вероятностные суждения для причинных, а не диагностических правил.

На рис. 14.3, б показано еще худшее упорядочение вершин: *MaryCalls*, *JohnCalls*, *Earthquake*, *Burglary*, *Alarm*. Для этой сети требуется задать 31 отдельную вероятность — точно такое же количество, как и при использовании полного совместного распределения. Однако важно понять, что любая из этих трех сетей может представлять точно такое же совместное распределение, как и другие. Просто в последних двух версиях не удалось успешно выявить все отношения условной независимости и поэтому вместо них пришлось ввести много ненужных чисел.

## Отношения условной независимости в байесовских сетях

В предыдущем разделе была определена “числовая” семантика для байесовских сетей в терминах представления полного совместного распределения, как в уравнении 14.1. Используя эту семантику для вывода метода составления байесовских сетей, мы пришли к заключению, что вершина является условно независимой от ее предшественников, если заданы ее родительские вершины. Как оказалось, можно также двигаться в другом направлении. Мы можем начать с “топологической” семантики, которая задает отношения условной независимости, закодированные в структуре графа, а из этой информации вывести “числовую” семантику. Топологическая семантика задается любой из приведенных ниже спецификаций, которые являются эквивалентными<sup>2</sup>.

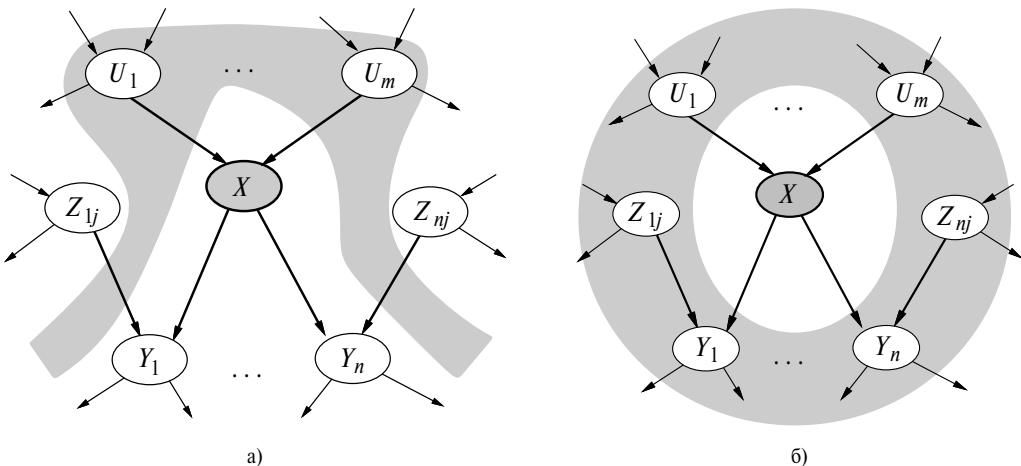
1. Вершина является условно независимой от вершин, не являющихся ее ~~потомками~~, если даны ее родительские вершины. Например, на рис. 14.2 вершина *JohnCalls* независима от *Burglary* и *Earthquake*, если дано значение *Alarm*.
2. Вершина является условно независимой от всех других вершин в сети, если даны ее родительские вершины, дочерние вершины и родительские вершины дочерних вершин, т.е. дано ее ~~марковское покрытие~~ (Markov blanket). Например, вершина *Burglary* независима от *JohnCalls* и *MaryCalls*, если даны *Alarm* и *Earthquake*.

Примеры применения этих спецификаций показаны на рис. 14.4. На основании приведенных утверждений об условной независимости и таблиц СРТ можно реконструировать полное совместное распределение; таким образом, “числовая” семантика и “топологическая” семантика являются эквивалентными.

### 14.3. ЭФФЕКТИВНОЕ ПРЕДСТАВЛЕНИЕ РАСПРЕДЕЛЕНИЙ УСЛОВНЫХ ВЕРОЯТНОСТЕЙ

Даже если максимальное количество родительских вершин  $k$  невелико, для заполнения таблицы СРТ, относящейся к некоторой вершине, необходимо будет задать вплоть до  $O(2^k)$  чисел, а также, возможно, потребуется значительный опыт оценки всех возможных обусловливающих случаев. К тому же на практике иногда встречается наихудшая ситуация, в которой связь между родительскими вершинами и дочерней вершиной является полностью произвольной. Обычно такие отношения можно описать с помощью ~~канонического распределения~~, которое соответствует некоторому стандартному образцу. В таких случаях всю таблицу можно задать, указав тип распределения и, возможно, введя несколько параметров, что намного проще по сравнению с определением экспоненциального количества параметров.

<sup>2</sup> Существует также общий топологический критерий, называемый **d-отделением**, для принятия решения о том, является ли множество вершин **X** независимым от другого множества **Y**, если задано третье множество **Z**. Этот критерий является довольно сложным и не требуется для разработки алгоритмов, приведенных в данной главе, поэтому мы его не описываем. Подробные сведения об этом критерии можно найти в [1191] или [1328], а в [1384] описан более интуитивно понятный метод проверки выполнения условий d-отделения.



*Рис. 14.4. Примеры использования спецификаций топологической семантики: вершина X является условно независимой от вершин, не являющихся ее потомками (например, от вершин  $Z_{i,j}$ ), если даны ее родительские вершины (вершины  $U_i$ , показанные в затененной области) (а); вершина X является условно независимой от всех других вершин в сети, если дано ее марковское покрытие (затененная область) (б)*

С другой стороны, примером простейшей ситуации является наличие **детерминированных вершин**. Детерминированная вершина характеризуется тем, что ее значение точно определяется значениями ее родительских вершин, без какой-либо неопределенности. Отношение между вершинами может быть логическим, например, отношение между родительскими вершинами *Canadian*, *US*, *Mexican* и дочерней вершиной *NorthAmerican* состоит в том, что значение дочерней вершины представляет собой дизьюнкцию значений родительских вершин. Такое отношение может также быть числовым, например, если значениями родительских вершин являются цены на конкретную модель автомобиля у нескольких дилеров, а значением дочерней вершины является цена, которую в конечном итоге заплатит охотник за низкими ценами, то значением дочерней вершины является минимальное из значений родительских вершин. Если же значениями родительских вершин являются притоки воды (через реки, ключи, росу) в озеро и оттоки воды (через реки, испарения, дренаж) из озера, а значением дочерней вершины является изменение уровня воды в озере, то значение дочерней вершины представляет собой общую разницу между родительскими вершинами с притоком и родительскими вершинами с оттоком.

Неопределенные отношения можно также часто охарактеризовать с использованием так называемых “зашумленных” логических отношений. Стандартным примером является отношение ~~и~~ зашумленного OR, которое представляет собой обобщение логического отношения OR. В пропозициональной логике можно составить утверждение, что высказывание *Fever* (Жар) является истинным тогда и только тогда, когда истинны высказывания *Cold* (Простуда), *Flu* (Грипп) или *Malaria* (Малария). Модель зашумленного OR позволяет учитывать неопределенность знаний в отношении способности каждой из родительских вершин вызывать присваивание истинного значения дочерней вершине, поскольку причинная связь между родительской и дочерней вершинами может быть заблокирована и поэтому иногда

пациент бывает простужен, но у него не обнаруживается жар. В этой модели приняты два предположения. Во-первых, в ней предполагается, что учтены все возможные причины. (Это — не такое строгое требование, как кажется на первый взгляд, поскольку всегда есть возможность ввести так называемую **вершину утечки** (leak node), которая покрывает “прочие причины”.) Во-вторых, предполагается, что блокирование каждой родительской вершины не зависит от блокирования любых других родительских вершин, например, та причина, которая блокирует возникновение жара под влиянием действия вершины *Malaria*, не зависит от тех причин, которые блокируют возникновение жара под действием вершины *Flu*. С учетом этих предположений значение вершины *Fever* является ложным тогда и только тогда, когда заблокированы все ее родительские вершины, имеющие истинное значение, а вероятность такой ситуации представляет собой произведение вероятностей блокировки каждой родительской вершины. Предположим, что такие отдельные вероятности блокировки выражаются следующим образом:

$$\begin{aligned} P(\neg\text{fever} | \text{cold}, \neg\text{flu}, \neg\text{malaria}) &= 0.6 \\ P(\neg\text{fever} | \neg\text{cold}, \text{flu}, \neg\text{malaria}) &= 0.2 \\ P(\neg\text{fever} | \neg\text{cold}, \neg\text{flu}, \text{malaria}) &= 0.1 \end{aligned}$$

В таком случае появляется возможность составить всю таблицу СРТ на основании этой информации и предположений модели зашумленного OR. Пример вычисления значений соответствующих элементов показан в табл. 14.1.

**Таблица 14.1. Пример вычисления значений таблицы СРТ**

<i>Cold</i>	<i>Flu</i>	<i>Malaria</i>	<i>P(Fever)</i>	<i>P(\neg Fever)</i>
<i>F</i>	<i>F</i>	<i>F</i>	0.0	1.0
<i>F</i>	<i>F</i>	<i>T</i>	0.9	<b>0.1</b>
<i>F</i>	<i>T</i>	<i>F</i>	0.8	<b>0.2</b>
<i>F</i>	<i>T</i>	<i>T</i>	0.98	$0.02 = 0.2 \times 0.1$
<i>T</i>	<i>F</i>	<i>F</i>	0.4	<b>0.6</b>
<i>T</i>	<i>F</i>	<i>T</i>	0.94	$0.06 = 0.6 \times 0.1$
<i>T</i>	<i>T</i>	<i>F</i>	0.88	$0.12 = 0.6 \times 0.2$
<i>T</i>	<i>T</i>	<i>T</i>	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

Вообще говоря, зашумленные логические отношения, в которых некоторая переменная зависит от  $k$  родительских переменных, могут быть описаны с использованием  $O(k)$  параметров вместо  $O(2^k)$  параметров, которые требуются для полной таблицы условных вероятностей. Благодаря этому задачи присваивания значений и обучения намного упрощаются. Например, распределения зашумленного OR и зашумленного MAX успешно использовались в сети CPCS [1232] для моделирования отношений между заболеваниями и симптомами в диагностике внутренних органов. При наличии 448 вершин и 906 связей в этой сети потребовалось только 8254 значения вместо 133 931 430 значений для сети с полными таблицами СРТ.

### Байесовские сети с непрерывными переменными

Для решения многих реальных задач требуется учитывать непрерывные количества, такие как уровень, масса, температура и сумма денежных средств; в действии-

тельности большая часть такого научного направления, как статистика, посвящена исследованию случайных переменных, области определения которых являются непрерывными. По определению непрерывные переменные имеют бесконечное количество возможных значений, поэтому невозможно явно задать условные вероятности для каждого значения. Один из возможных способов обработки непрерывных переменных состоит в избавлении от них с помощью **дискретизации**, т.е. разделения всех возможных значений на постоянное множество интервалов. Например, температуры могут быть разделены на интервалы ( $<0^{\circ}\text{C}$ ), ( $0^{\circ}\text{C}-100^{\circ}\text{C}$ ) и ( $>100^{\circ}\text{C}$ ). Иногда дискретизация становится вполне адекватным решением, но часто приводит к значительной потере точности и появлению очень больших таблиц СРТ. Еще одно решение состоит в определении стандартных семейств функций плотности вероятностей (см. приложение А), которые задаются с помощью конечного количества **параметров**. Например, гауссово (или нормальное) распределение  $N(\mu, \sigma^2)$  ( $x$ ) имеет в качестве параметров среднее  $\mu$  и дисперсию  $\sigma^2$ .

Сеть, в которой имеются и дискретные, и непрерывные переменные, называется **гибридной байесовской сетью**. Для составления гибридной сети необходимо определить два новых типа распределений: условное распределение для непрерывной переменной, если даны дискретные или непрерывные родительские переменные, и условное распределение для дискретной переменной, если даны непрерывные родительские переменные. Рассмотрим простой пример, приведенный на рис. 14.5, в котором клиент покупает какие-то фрукты в зависимости от их стоимости, которая, в свою очередь, зависит от размера урожая и от того, применяется ли правительенная программа субсидий. Переменная *Cost* (Стоимость) является непрерывной и имеет непрерывные и дискретные родительские переменные; переменная *Buys* (Покупает) является дискретной и имеет непрерывную родительскую переменную.

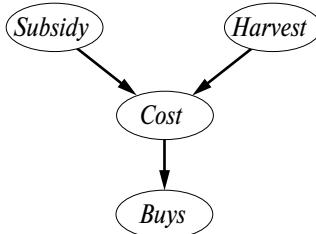


Рис. 14.5. Простая сеть с дискретными переменными (*Subsidy* — субсидия) и *Buys* — покупает) и непрерывными переменными (*Harvest* — урожай и *Cost* — стоимость)

Для переменной *Cost* необходимо задать распределение  $P(Cost | Harvest, Subsidy)$ . Это дискретное родительское значение учитывается с помощью явного перечисления, т.е. определения и значения  $P(Cost | Harvest, subsidy)$ , и значения  $P(Cost | Harvest, \neg subsidy)$ . Чтобы учесть переменную *Harvest*, требуется определить, как распределение по стоимости зависит от непрерывного значения *h* переменной *Harvest*. Иными словами, необходимо определить параметры распределения стоимости как функции от *h*. Наиболее часто применяемым вариантом является **линейное гауссово распределение**, в котором дочерняя переменная имеет

гауссового распределение со средним  $\mu$ , изменяющимся линейно в зависимости от значения родительской переменной, и с постоянным среднеквадратичным отклонением  $\sigma$ . Необходимо иметь два распределения, одно для  $subsidy$ , а другое для  $\neg subsidy$ , с разными параметрами:

$$P(c | h, subsidy) = N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{c - (a_t h + b_t)}{\sigma_t} \right)^2}$$

$$P(c | h, \neg subsidy) = N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{c - (a_f h + b_f)}{\sigma_f} \right)^2}$$

Итак, в данном примере условное распределение для переменной  $Cost$  было задано путем обозначения его как линейного гауссова распределения и предоставления параметров  $a_t, b_t, \sigma_t$ ,  $a_f, b_f$  и  $\sigma_f$ . Эти два отношения показаны на рис. 14.6, а и б. Обратите внимание на то, что в каждом случае наклон кривых является отрицательным, поскольку цена уменьшается по мере увеличения поставок. (Безусловно, из предположения о линейности следует, что цена в некоторый момент станет отрицательной; линейная модель является приемлемой, только если объем урожая ограничен каким-то узким диапазоном.) На рис. 14.6, в показано распределение  $P(c | h)$ , усредненное по двум возможным значениям  $Subsidy$ , в предположении, что каждое из этих двух значений имеет априорную вероятность 0,5. Данный пример показывает, что даже с помощью очень простых моделей могут быть представлены весьма интересные распределения.

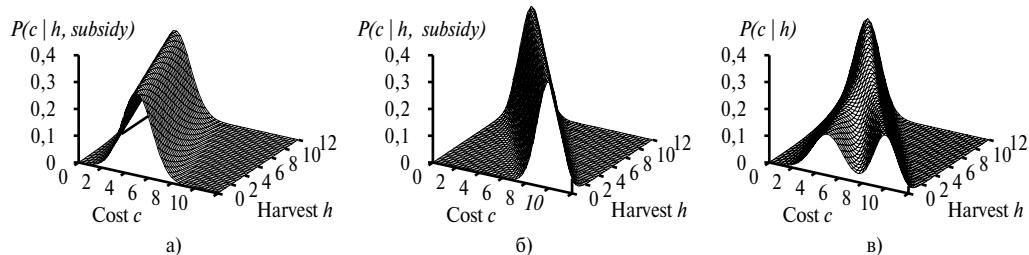


Рис. 14.6. Графики отношений. На графиках а) и б) показаны распределения вероятностей по значению стоимости  $Cost$  как функции от объема урожая  $Harvest$ , при условии, что переменная  $Subsidy$  имеет соответственно истинное и ложное значения. На графике в) показано распределение  $P(Cost | Harvest)$ , полученное путем суммирования по двум случаям, связанным с субсидией (предоставление и не предоставление)

Линейное гауссово условное распределение обладает некоторыми особыми свойствами. Сеть, содержащая только непрерывные переменные с линейными гауссовыми распределениями, имеет совместное распределение, представляющее собой многомерное гауссово распределение по всем переменным<sup>3</sup> (упр. 14.5).

<sup>3</sup> Из этого следует, что вероятностный вывод в линейных гауссовых сетях в наихудшем случае требует  $O(n^3)$  времени независимо от топологии сети. В разделе 14.4 будет показано, что вероятностный вывод в сетях с дискретными переменными является NP-трудным.

(Многомерное гауссово распределение представляет собой поверхность, заданную в нескольких измерениях, которая имеет пик в районе среднего значения, в  $n$  измерениях, и уменьшается по всем направлениям.) Если в сеть введены дискретные переменные (при условии, что ни одна дискретная переменная не является дочерней применительно к непрерывной переменной), то сеть определяет **условное гауссово распределение**, или распределение CG (conditional Gaussian): если дано любое присваивание дискретным переменным, то распределение по непрерывным переменным становится многомерным гауссовым распределением.

Теперь обратимся к распределениям для дискретных переменных с непрерывными родительскими переменными. Например, рассмотрим вершину  $Buys$  на рис. 14.5. Представляется обоснованным предположение, что клиент сделает покупку, если стоимость является низкой, и не сделает покупку, если она высока, а также, что вероятность покупки изменяется плавно в некотором промежуточном регионе. Другими словами, условное распределение напоминает “мягкую” пороговую функцию. Один из способов определения мягких пороговых функций состоит в использовании интеграла стандартного нормального распределения:

$$\Phi(x) = \int_{-\infty}^x N(0, 1)(x) dx$$

В таком случае вероятность события  $Buys$ , если дано значение  $Cost$ , может выражаться следующей формулой:

$$P(buys | Cost=c) = \Phi((-c+\mu)/\sigma)$$

которая означает, что пороговое значение стоимости обнаруживается в районе  $\mu$ , ширина порогового региона пропорциональна  $\sigma$ , а вероятность покупки уменьшается по мере возрастания стоимости.

Такое распределение вероятностей называется **пробит-распределением** (probit distribution) и показано на рис. 14.7, а. Возможность применения распределения с такой формой может быть обосновано тем, что соответствующий процесс принятия решения имеет твердый порог, но точное расположение этого порогового значения подвержено воздействию случайного гауссова шума. Альтернативным по отношению к пробит-распределению является **логит-распределение** (logit distribution), в котором для формирования мягкого порога используется **сигмоидальная функция** (sigmoid function):

$$P(buys | Cost=c) = \frac{1}{1 + \exp(-2 \frac{-c+\mu}{\sigma})}$$

Это распределение показано на рис. 14.7, б. Два распределения внешне выглядят одинаковыми, но фактически логит-распределения имеют более длинные “хвосты”. Пробит-распределения часто лучше подходят в реальных ситуациях, а логит-распределения иногда проще поддаются математической обработке. Последние широко используются в нейронных сетях (глава 20). И пробит- и логит-распределения могут быть обобщены для учета многочисленных непрерывных родительских значений путем применения линейной комбинации родительских значений. Расширения для случая многомерных дискретных дочерних значений рассматриваются в упр. 14.6.

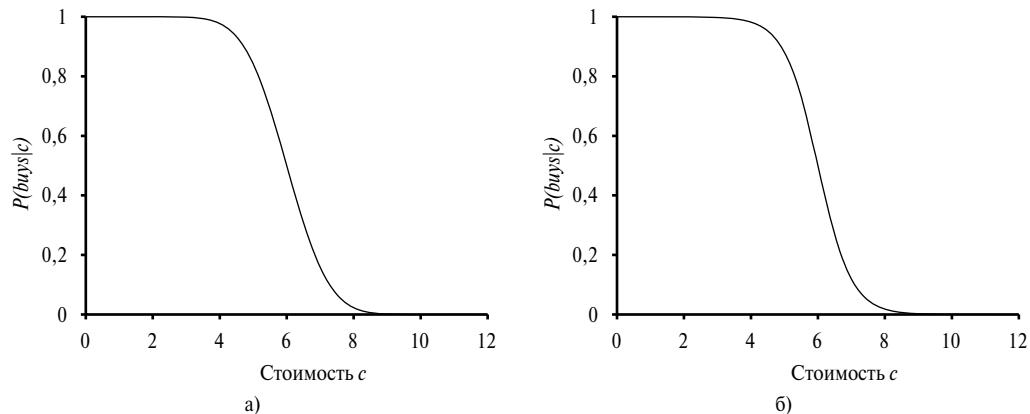


Рис. 14.7. Примеры других распределений: пробит-распределение для вероятности события Buys, если дано значение Cost, которое характеризуется значениями  $\mu = 6.0$  и  $\sigma = 1.0$  (а); логит-распределение с такими же параметрами (б)

#### 14.4. ТОЧНЫЙ ВЕРОЯТНОСТНЫЙ ВЫВОД В БАЙЕСОВСКИХ СЕТЯХ

Основной задачей для любой системы вероятностного вывода является вычисление распределения апостериорных вероятностей для множества **переменных запроса**, если дано некоторое наблюдаемое  $\mathbf{событие}$ , т.е. если выполнено некоторое присваивание значений множеству **переменных свидетельства**. Мы будем использовать систему обозначений, введенную в главе 13:  $X$  обозначает переменную запроса;  $\mathbf{E}$  — множество переменных свидетельства,  $E_1, \dots, E_m$ ;  $\mathbf{e}$  — конкретное наблюдаемое событие;  $\mathbf{Y}$  обозначает переменные, отличные от переменных свидетельства,  $Y_1, \dots, Y_n$  (иногда называемые  $\mathbf{скрытыми переменными}$ ). Таким образом, полное множество переменных определяется выражением  $\mathbf{x} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$ . В типичном запросе<sup>4</sup> содержится просьба определить распределение апостериорных вероятностей  $\mathbf{P}(X|\mathbf{e})$ .

В сети с описанием взлома может наблюдаться событие, в котором  $JohnCalls=true$  и  $MaryCalls=true$ . В таком случае можно ввести следующий запрос, скажем, для определения вероятности того, что произошел взлом:

$$\mathbf{P}(\text{Burglary} | JohnCalls=true, MaryCalls=true) = <0.284, 0.716>$$

В этом разделе рассматриваются точные алгоритмы вычисления апостериорных вероятностей и приводятся сведения о сложности такой задачи. Как оказалось, в общем случае эта задача неразрешима, поэтому в разделе 14.5 рассматриваются методы приближенного вероятностного вывода.

<sup>4</sup> Предполагается, что переменная запроса не относится к числу переменных свидетельства; если бы она была таковой, то в распределении апостериорных вероятностей для  $X$  была бы присвоена вероятность 1 этому наблюдаемому значению. Кроме того, для упрощения предполагается, что запрос касается только единственной переменной. Приведенные здесь алгоритмы могут быть легко дополнены для обработки запросов, касающихся нескольких переменных.

## Вероятностный вывод с помощью перебора

Как говорилось в главе 13, любую условную вероятность можно вычислить, суммируя элементы из полного совместного распределения. Более конкретно следует указать, что ответ на запрос  $P(X|e)$  можно получить с использованием уравнения 13.6, которое мы повторяем здесь для удобства:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

Итак, любая байесовская сеть создает исчерпывающее представление полного совместного распределения. А именно, в уравнении 14.1 показано, что термы  $P(x, e, y)$  в совместном распределении можно записать в виде произведений условных вероятностей, взятых из сети. Поэтому *на любой запрос можно найти ответ с помощью байесовской сети, вычисляя суммы произведений условных вероятностей из этой сети.*

В листинге 13.2 был приведен алгоритм `Enumerate-Joint-Ask`, обеспечивающий вероятностный вывод путем перебора элементов полного совместного распределения, который принимает на входе полное совместное распределение  $P$  и выполняет в нем поиск значений. Этот алгоритм несложно модифицировать так, чтобы он принимал на входе байесовскую сеть  $bn$  и “отыскивал” элементы совместного распределения, умножая соответствующие элементы таблиц СРТ, взятые из сети  $bn$ .

Рассмотрим запрос  $P(Burglary|JohnCalls=true, MaryCalls=true)$ . Скрытыми переменными для этого запроса являются *Earthquake* и *Alarm*. Из уравнения 13.6, используя начальные символы имен переменных для сокращения длины выражений, можно получить следующее<sup>5</sup>:

$$P(B|j, m) = \alpha P(B, j, m) = \alpha \sum_e \sum_a P(B, e, a, j, m)$$

В таком случае выражение в терминах элементов таблицы СРТ может быть получено с учетом семантики байесовских сетей (уравнение 14.1). Для упрощения получим соответствующее выражение только для случая *Burglary = true*:

$$P(b|j, m) = \alpha \sum_e \sum_a P(b) P(e) P(a|b, e) P(j|a) P(m|a)$$

Для вычисления этого выражения необходимо сложить четыре терма, каждый из которых вычисляется путем умножения пяти чисел. В наихудшем случае, когда приходится находить сумму почти по всем переменным, сложность этого алгоритма для сети с  $n$  булевыми переменными равна  $O(n2^n)$ .

Некоторое упрощение может быть достигнуто на основе следующих простых наблюдений: терм  $P(b)$  представляет собой константу и может быть перенесен за пре-

<sup>5</sup> Такое выражение, как  $\sum_e P(a, e)$ , обозначает сумму  $P(A=a, E=e)$  по всем возможным значениям  $e$ . При этом может возникать неоднозначность, связанная с тем, что  $P(e)$  используется для обозначения и  $P(E=true)$ , и  $P(E=e)$ , но из контекста должно быть ясно, какой смысл имеет это выражение; в частности, в контексте суммы подразумевается последнее значение.

делы выражений суммирования по  $a$  и  $e$ , а терм  $P(e)$  — перенесен за пределы выражения суммирования по  $a$ . Поэтому получаем следующее:

$$P(b|j,m) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(j|a) P(m|a) \quad (14.3)$$

Это выражение можно вычислить, последовательно обрабатывая в цикле его переменные и перемножая в ходе этого элементы таблицы СРТ. При каждом суммировании необходимо также выполнить цикл по возможным значениям переменной. Структура этих вычислений показана на рис. 14.8. Используя числа, приведенные на рис. 14.2, получим выражение  $P(b|j,m) = \alpha \times 0.00059224$ . Соответствующее вычисление для  $\neg b$  приводит к получению выражения  $\alpha \times 0.0014919$ , поэтому имеет место следующее:

$$P(B|j,m) = \alpha <0.00059224, 0.0014919> \approx <0.284, 0.716>$$

Таким образом, вероятность взлома, при условии, что поступили звонки от обоих соседей, составляет около 28%.

Процесс вычисления выражения, приведенного в уравнении 14.3, показан в виде дерева синтаксического анализа выражения на рис. 14.8. В алгоритме Enumeration-Ask, приведенном в листинге 14.1, вычисление подобных деревьев осуществляется с использованием рекурсии в глубину. Таким образом, пространственная сложность алгоритма Enumeration-Ask зависит от количества переменных только линейно; по сути этот алгоритм вычисляет суммы по полному совместному распределению, даже не формируя его явно. К сожалению, его временная сложность для сети с  $n$  булевыми переменными всегда составляет  $O(2^n)$ ; это лучше по сравнению с оценкой  $O(n2^n)$  для простого подхода, описанного выше, но все еще довольно велика. В отношении дерева, приведенного на рис. 14.8, необходимо сделать еще одно замечание — в нем явно показаны повторяющиеся подвыражения, вычисляемые с помощью этого алгоритма. Произведения  $P(j|a) P(m|a)$  и  $P(j|\neg a) P(m|\neg a)$  вычисляются дважды, по одному для каждого значения  $e$ . В следующем разделе описан общий метод, позволяющий избежать таких избыточных вычислений.

#### Листинг 14.1. Алгоритм перебора для получения ответов на запросы в байесовских сетях

```

function Enumeration-Ask( $X, e, bn$ ) returns распределение по  $X$ 
  inputs:  $X$ , переменная запроса
     $e$ , наблюдаемые значения переменных  $E$ 
     $bn$ , байесовская сеть с переменными
       $\{X\} \cup E \cup Y\}$  /*  $Y$  — скрытые переменные */

   $Q(X) \leftarrow$  распределение по  $X$ , первоначально пустое
  for each значение  $x_i$  переменной  $X$  do
    дополнить  $e$  значением  $x_i$  переменной  $X$ 
     $Q(x_i) \leftarrow$  Enumerate-All( $Vars[bn], e$ )
  return Normalize( $Q(X)$ )

function Enumerate-All( $vars, e$ ) returns действительное число
  if Empty?( $vars$ ) then return 1.0
   $Y \leftarrow First(vars)$ 
  if переменная  $Y$  имеет значение  $u$  в множестве  $e$ 

```

```

then return  $P(Y|parents(Y)) \times \text{Enumerate-All}(\text{Rest}(vars), e)$ 
else return  $\sum_y P(y|parents(Y)) \times \text{Enumerate-All}(\text{Rest}(vars), e_y),$ 

```

где  $e_y$  представляет собой множество  $e$ , дополненное значением  $Y = y$

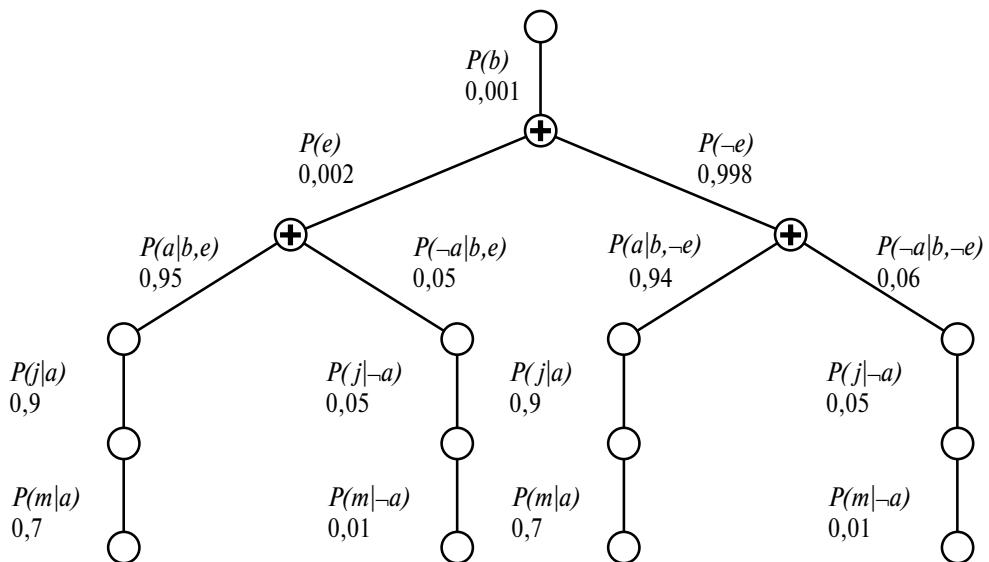


Рис. 14.8. Структура выражения, показанного в уравнении 14.3. Процесс вычисления осуществляется сверху вниз; при этом значения вдоль каждого пути умножаются и суммируются в узлах “+”. Обратите внимание на то, что пути для  $j$  и  $m$  повторяются

### Алгоритм устранения переменной

Описанный выше алгоритм перебора можно существенно улучшить, исключив повторные вычисления такого типа, как показано на рис. 14.8. Его идея проста: выполнить вычисление один раз и сохранить результаты для дальнейшего использования. В этом выражается одна из форм динамического программирования. Существует несколько версий такого подхода; в данной главе будет представлен алгоритм **устраниния переменной** (variable elimination), который является наиболее простым. Устранение переменной осуществляется путем вычисления выражений, подобных представленному в уравнении 14.3, в порядке справа налево (т.е. на рис. 14.8 — сверху вниз). Промежуточные результаты сохраняются, а операции суммирования по каждой переменной выполняются только для тех частей выражения, которые зависят от этой переменной.

Проиллюстрируем этот процесс на примере сети с описанием взлома. Нам необходимо вычислить следующее выражение:

$$\mathbf{P}(B|j,m) = \alpha \underbrace{\mathbf{P}(B)}_B \sum_e \underbrace{P(e)}_E \sum_a \underbrace{\mathbf{P}(a|B,e)}_A \underbrace{P(j|a)}_J \underbrace{P(m|a)}_M$$

Обратите внимание на то, что каждая часть этого выражения аннотирована именем связанной с ней переменной; эти части называются **факторами**. Этапы работы алгоритма устранения переменной описаны ниже.

- Фактор для  $M$ ,  $P(m|a)$ , не требует суммирования по  $M$  (поскольку значение  $M$  уже фиксировано). Мы сохраним эту вероятность при условии, что задано каждое значение  $a$  в виде следующего двухэлементного вектора:

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m|a) \\ P(m|\neg a) \end{pmatrix}$$

(Обозначение  $\mathbf{f}_M$  показывает, что для получения  $\mathbf{f}$  используется  $M$ .)

- Аналогичным образом сохраняется фактор для  $J$  в виде двухэлементного вектора  $\mathbf{f}_J(A)$ .
- Фактор для  $A$  выражается распределением  $P(a|B, e)$ , которое представляет собой матрицу  $\mathbf{f}_A(A, B, E)$  с размерами  $2 \times 2 \times 2$ .
- Теперь необходимо устраниТЬ путем суммирования из произведения этих трех факторов переменную  $A$ , что позволит получить матрицу  $2 \times 2$ , индексы которой пробегают только по  $B$  и  $E$ . Поместим штрих над  $A$  в имени этой матрицы для указания на то, что  $A$  устраниена путем суммирования:

$$\begin{aligned} \mathbf{f}_{AJM}(B, E) &= \sum_a \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &= \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) + \mathbf{f}_A(\neg a, B, E) \times \mathbf{f}_J(\neg a) \times \mathbf{f}_M(\neg a) \end{aligned}$$

Применяемый в этом выражении процесс умножения называется процессом получения **точечного произведения** (pointwise product) и будет вскоре описан.

- Обрабатаем таким же образом переменную  $E$ : исключим  $E$  путем суммирования из произведения  $\mathbf{f}_E(E)$  и  $\mathbf{f}_{AJM}(B, E)$ :

$$\mathbf{f}_{EAJM}(B) = \mathbf{f}_E(e) \times \mathbf{f}_{AJM}(B, e) + \mathbf{f}_E(\neg e) \times \mathbf{f}_{AJM}(B, \neg e)$$

- Теперь мы можем вычислить ответ, умножив фактор для  $B$  (т.е.  $\mathbf{f}_B(B) = P(B)$ ) на накопленную матрицу  $\mathbf{f}_{EAJM}(B)$ :

$$P(B|j, m) = \alpha \mathbf{f}_B(B) \times \mathbf{f}_{EAJM}(B)$$

В упр. 14.7, а предлагается проверить, действительно ли данный процесс приводит к получению правильного ответа.

Исследуя приведенную выше последовательность этапов, можно убедиться в том, что требуются две основные вычислительные операции: получение точечного произведения пары факторов и исключение некоторой переменной путем суммирования из произведения факторов.

Точечное произведение — это не результат умножения матриц и не результат поэлементного умножения. Точечное произведение двух факторов,  $\mathbf{f}_1$  и  $\mathbf{f}_2$ , представляет собой новый фактор  $\mathbf{f}$ , переменные которого являются объединением переменных из  $\mathbf{f}_1$  и  $\mathbf{f}_2$ . Предположим, что два фактора имеют общие переменные  $Y_1, \dots, Y_k$ . В таком случае получим следующее:

$$\mathbf{f}(X_1 \dots X_j, Y_1 \dots Y_k, Z_1 \dots Z_l) = \mathbf{f}_1(X_1 \dots X_j, Y_1 \dots Y_k) \mathbf{f}_2(Y_1 \dots Y_k, Z_1 \dots Z_l)$$

Если все переменные являются бинарными, то факторы  $\mathbf{f}_1$  и  $\mathbf{f}_2$  имеют  $2^{j+k}$  и  $2^{k+1}$  элементов соответственно, а точечное произведение включает  $2^{j+k+1}$  элементов. Например, если даны два фактора,  $\mathbf{f}_1(A, B)$  и  $\mathbf{f}_2(B, C)$ , с распределениями вероятностей, показанными ниже, то точечное произведение  $\mathbf{f}_1 \times \mathbf{f}_2$  задается в табл. 14.2 как  $\mathbf{f}_3(A, B, C)$ .

Таблица 14.2. Пример произведения факторов

<b>A</b>	<b>B</b>	$f_1(A, B)$	<b>B</b>	<b>C</b>	$f_2(B, C)$	<b>A</b>	<b>B</b>	<b>C</b>	$f_3(A, B, C)$
T	T	.3	T	T	.2	T	T	T	.3 × .2
T	F	.7	T	F	.8	T	T	F	.3 × .8
F	T	.9	F	T	.6	T	F	T	.7 × .6
F	F	.1	F	F	.4	T	F	F	.7 × .4
						F	T	T	.9 × .2
						F	T	F	.9 × .8
						F	F	T	.1 × .6
						F	F	F	.1 × .4

Операция исключения некоторой переменной путем суммирования из произведения факторов также является несложной. Единственное затруднение заключается в следующем: в этой операции необходимо учитывать, что любой фактор, который не зависит от переменной, подлежащей исключению путем суммирования, может быть вынесен за пределы выражения суммирования, например, как показано ниже.

$$\sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) \times \mathbf{f}_J(A) \times \mathbf{f}_M(A) = \\ \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e)$$

Теперь вычисляется точечное произведение в выражении суммирования и переменная исключается путем суммирования из результирующей матрицы следующим образом:

$$\mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \mathbf{f}_{E,A}(A, B)$$

Обратите внимание на то, что матрицы не умножаются до тех пор, пока не потребуется исключить некоторую переменную путем суммирования из накопленного произведения. В данный момент умножаются только те матрицы, которые включают переменную, подлежащую исключению путем суммирования. Если даны процедуры получения точечного произведения и исключения путем суммирования, то сам алгоритм устранения переменной может быть записан весьма просто, как показано в листинге 14.2.

**Листинг 14.2. Алгоритм устранения переменной, предназначенный для получения ответов на запросы в байесовских сетях**

---

```

function Elimination-Ask( $X, e, bn$ ) returns распределение по  $X$ 
  inputs:  $X$ , переменная запроса
     $e$ , свидетельство, определяемое как некоторое событие
     $bn$ , байесовская сеть, которая задает совместное
      распределение  $P(X_1, \dots, X_n)$ 

  factors  $\leftarrow []$ ; vars  $\leftarrow$  Reverse(Vars[ $bn$ ])
  for each var in vars do
    factors  $\leftarrow$  [Make-Factor(var,  $e$ ) | factors]
    if переменная var является скрытой
      then factors  $\leftarrow$  Sum-Out(var, factors)
  return Normalize(Pointwise-Product(factors))

```

---

Рассмотрим еще один запрос:  $P(JohnCalls | Burglary=true)$ . Как обычно, на первом этапе необходимо записать вложенное выражение суммирования:

$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b, e) P(J|a) \sum_m P(m|a)$$

Если бы мы вычисляли это выражение справа налево, то заметили бы нечто интересное: терм  $\sum_m P(m|a)$  равен 1 по определению! Поэтому сразу отпадает необходимость учитывать это выражение; переменная  $M$  не имеет отношения к данному запросу. Этую мысль можно выразить иным образом: результат запроса  $P(JohnCalls | Burglary=true)$  не изменится, если из сети вообще будет исключена переменная *MaryCalls*. Вообще говоря, из сети может быть удалена любая листовая вершина, если она не относится к переменной запроса или к переменной свидетельства. После ее удаления могут еще оставаться некоторые листовые вершины, которые также могут не относиться к данному запросу. Продолжая указанный процесс, мы в конечном итоге обнаружим, что *к запросу не относится любая переменная, которая не является потомком переменной запроса или переменной свидетельства*. Поэтому алгоритм устранения переменной позволяет удалять все эти переменные, прежде чем приступать к вычислению ответа на запрос.

### Сложность точного вероятностного вывода

Выше было показано, что устранение переменной является более эффективным по сравнению с перебором, поскольку позволяет предотвратить повторяющиеся вычисления (а также обеспечить удаление не относящихся к делу переменных). Требования процесса устранения переменной к времени и пространству зависят в основном от размера наибольшего фактора, сформированного в процессе работы данного алгоритма. А этот размер, в свою очередь, зависит от порядка устранения переменных и от структуры сети.

Сеть с описанием взлома, приведенная на рис. 14.2, принадлежит к семейству сетей, в которых имеется самое большее один неориентированный путь между любыми двумя вершинами в сети. Такие сети называются **односвязными** (singly connected) сетями, или **полидеревьями** (polytree), и обладают одним особо привле-

кательным свойством:  $\Leftrightarrow$  временная и пространственная сложность точного вероятностного вывода в полидеревьях линейно зависят от размера сети. В данном случае размер определяется как количество элементов таблиц СРГ; если количество родительских вершин каждой вершины ограничено некоторой константой, то сложность также будет зависеть линейно от количества вершин. Эти результаты справедливы для любого упорядочения, совместимого с топологическим упорядочением сети (упр. 14.7).

Для  $\bowtie$  многосвязных (multiply connected) сетей, подобных приведенной на рис. 14.9, а, процедура устранения переменной в наихудшем случае может иметь экспоненциальную временную и пространственную сложность, даже если количество родительских вершин в расчете на каждую вершину ограничено. В этом нет ничего удивительного, если учесть, что  $\Leftrightarrow$  вероятностный вывод в байесовских сетях является NP-трудным, поскольку включает вывод в пропозициональной логике как частный случай. Фактически можно показать (упр. 14.8), что эта задача является настолько трудной, насколько трудна задача вычисления количества выполняющих присваиваний для формулы пропозициональной логики. Это означает, что данная задача является #P-трудной (читается “шарп-Р-трудной”, или “диэз-Р-трудной”), т.е. строго труднее, чем NP-полные задачи.

Существует тесная связь между сложностью вероятностного вывода в байесовской сети и сложностью задач удовлетворения ограничений (Constraint Satisfaction Problem — CSP). Как было показано в главе 5, трудность решения дискретной задачи CSP зависит от того, насколько “древовидным” является ее граф ограничений. Такие показатели, как **ширина гипердерева**, которые устанавливают пределы сложности решения задачи CSP, могут также применяться непосредственно к байесовским сетям. Более того, существует возможность обобщить алгоритм устранения переменной таким образом, чтобы он позволял находить решения не только в байесовских сетях, но и в задачах CSP.

## Алгоритмы кластеризации

Описанный выше алгоритм устранения переменной является простым и эффективным средством получения ответов на отдельные запросы. Если же возникает необходимость вычислить апостериорные вероятности для всех переменных в сети, то этот алгоритм может оказаться менее эффективным. Например, в сети с полидревовидной структурой для этого потребуется выдать  $O(n)$  запросов со стоимостью  $O(n)$  каждый, что в сумме составляет затраты времени  $O(n^2)$ . Использование алгоритмов  $\bowtie$  кластеризации (известных также под названием алгоритмов  $\bowtie$  дерева соединения — join tree), позволяет сократить это время до  $O(n)$ . По указанной причине данные алгоритмы широко используются в коммерческих инструментальных средствах на основе байесовских сетей.

Основная идея кластеризации состоит в том, что отдельные узлы в дереве должны быть соединены для формирования кластерных вершин таким образом, чтобы результирующая сеть стала полидеревом. Например, многосвязная сеть, показанная на рис. 14.9, а, может быть преобразована в полидрево путем объединения вершин *Sprinkler* (Опрыскиватель) и *Rain* (Дождь) в кластерную вершину, называемую *Sprinkler+Rain* (см. рис. 14.9, б). Эти две булевые вершины заменяются мегавершиной, которая принимает четыре возможных значения: *TT*, *TF*, *FT* и *FF*. Данная

мегавершина имеет только одну родительскую вершину — булеву переменную *Cloudy*, поэтому для нее количество обусловливающих случаев равно двум.

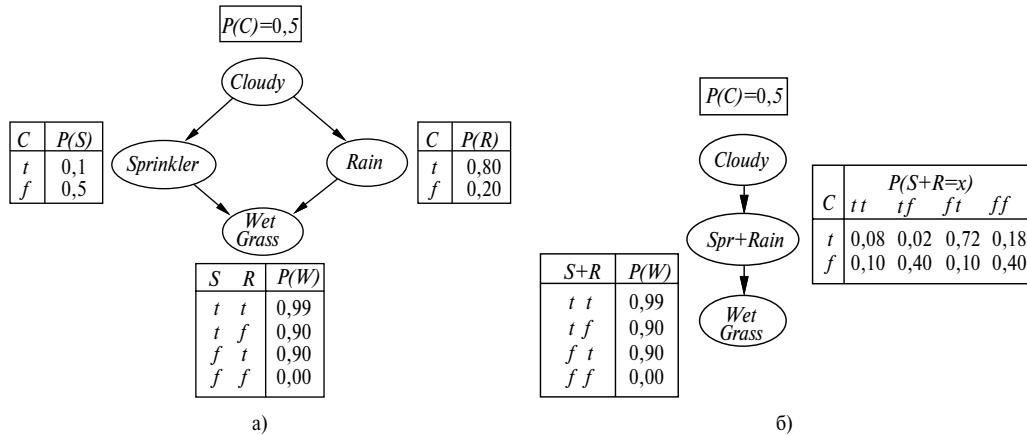


Рис. 14.9. Пример применения кластеризации: многосвязная сеть с таблицами условных вероятностей (а); кластеризованный эквивалент многосвязной сети (б)

После преобразования сети в форму полидерева применяется алгоритм вероятностного вывода специального назначения. По сути этот алгоритм представляет собой одну из форм алгоритма распространения ограничений (см. главу 5), где ограничения обеспечивают согласование соседних кластеров по апостериорной вероятности любых переменных, которые являются в них общими. При наличии тщательно продуманных средств учета этот алгоритм позволяет вычислять апостериорные вероятности для всех вершин в сети, отличных от вершин свидетельства, за время  $O(n)$ , где  $n$  теперь обозначает размер модифицированной сети. Тем не менее NP-трудность решаемой задачи не исчезает: если сеть требует экспоненциальных затрат времени и пространства при устраниении переменной, то экспоненциальные затраты времени и пространства требуются и для построения таблиц СРТ в кластеризованной сети.

## 14.5. ПРИБЛИЖЕННЫЙ ВЕРОЯТНОСТНЫЙ ВЫВОД В БАЙЕСОВСКИХ СЕТЯХ

Исходя из того, что точный вероятностный вывод в больших многосвязных сетях является неосуществимым, важно предусмотреть методы приближенного вероятностного вывода. В настоящем разделе описаны рандомизированные алгоритмы выборки (называемые также алгоритмами **Монте-Карло**), обеспечивающие получение приближенных ответов, точность которых зависит от количества сформированных выборок. В последние годы алгоритмы Монте-Карло нашли широкое распространение в компьютерных науках для получения оценочных значений величин, которые трудно вычислить точно. Например, алгоритм эмуляции отжига, описанный в главе 4, представляет собой метод Монте-Карло для задач оптимизации. В этом разделе нас интересует применение методов выборки для вычисления апо-

стериорных вероятностей. Здесь описаны два семейства алгоритмов: непосредственная выборка и выборка с помощью цепи Маркова. Два других подхода (вариационные методы и метод циклического распространения) упоминаются в библиографических и исторических заметках, приведенных в конце данной главы.

### Методы непосредственной выборки

Примитивной операцией в любом алгоритме выборки является формирование выборок из известного распределения вероятностей. Например, подлинная монета может рассматриваться как случайная переменная *Coin* со значениями *<heads, tails>* и априорным распределением  $P(Coin) = <0.5, 0.5>$ . Операция получения выборки из этого распределения полностью аналогична подбрасыванию монеты: с вероятностью 0,5 эта операция будет возвращать значение *heads* (орел) и с такой же вероятностью — значение *tails* (решка). Если имеется источник случайных чисел в диапазоне  $[0, 1]$ , сформировать любые распределения по одной переменной совсем несложно (см. упр. 14.9).

В процессе случайного формирования выборки простейшего типа для байесовских сетей, рассматриваемом в этом разделе, вырабатываются события, не имеющие связанных с ними свидетельств, которые могут быть зарегистрированы в сети. Идея этого метода состоит в том, что выборка должна формироваться последовательно по каждой переменной, в топологическом порядке. Распределение вероятностей, из которого берется выборка значения, обусловливается значениями, уже присвоенными родительским переменным этой переменной. Соответствующий алгоритм приведен в листинге 14.3. Проиллюстрируем его работу на примере сети, приведенной на рис. 14.9, *a*, предполагая, что имеет место упорядочение *[Cloudy, Sprinkler, Rain, WetGrass]*, как описано ниже.

#### Листинг 14.3. Алгоритм формирования выборки, который вырабатывает события на основании байесовской сети

---

```
function Prior-Sample(bn) returns событие, выработанное путем применения
    операции формирования выборки к априорному распределению,
    заданному в виде сети bn
inputs: bn, байесовская сеть, задающая совместное
        распределение  $P(X_1, \dots, X_n)$ 

x  $\leftarrow$  событие с n элементами
for i = 1 to n do
    xi  $\leftarrow$  случайная выборка из  $P(X_i | parents(X_i))$ 
return x
```

---

1. Выборка из распределения  $P(Cloudy) = <0.5, 0.5>$ ; предположим, что она возвращает *true*.
2. Выборка из распределения  $P(Sprinkler | Cloudy=true) = <0.1, 0.9>$ ; предположим, что она возвращает *false*.
3. Выборка из распределения  $P(Rain | Cloudy=true) = <0.8, 0.2>$ ; предположим, что она возвращает *true*.

4. Выборка из распределения  $P(WetGrass | Sprinkler=false, Rain=true) = <0.9, 0.1>$ ; предположим, что она возвращает *true*.

В данном случае алгоритм Prior-Sample возвращает событие  $[true, false, true, true]$ .

Можно легко показать, что алгоритм Prior-Sample формирует выборки на основе априорного совместного распределения, которое задано рассматриваемой сетью. Прежде всего предположим, что  $S_{PS}(x_1, \dots, x_n)$  представляет собой вероятность того, что конкретное событие сформировано алгоритмом Prior-Sample. Достаточно лишь проанализировать сам процесс формирования выборки, чтобы убедиться в справедливости следующего соотношения, поскольку каждый этап формирования выборки зависит только от значений родительских переменных:

$$S_{PS}(x_1 \dots x_n) = \prod_{i=1}^n P(x_i | parents(X_i))$$

Это выражение должно показаться читателю весьма знакомым, поскольку оно определяет также вероятность события в соответствии с представлением совместного распределения в байесовской сети, как указано в уравнении 14.1. Поэтому получаем следующее:

$$S_{PS}(x_1 \dots x_n) = P(x_1 \dots x_n)$$

Благодаря такому простому факту задача получения ответов на вопросы с помощью выборок решается очень просто.

В любом алгоритме формирования выборки ответы вычисляются путем подсчета фактически сформированных выборок. Предположим, что общее количество выборок равно  $N$ , а также допустим, что  $N(x_1, \dots, x_n)$  — частота конкретного события  $x_1, \dots, x_n$ . Следует полагать, что эта частота сойдется в пределе к ее ожидаемому значению, соответствующему вероятности сформированной выборки:

$$\lim_{N \rightarrow \infty} \frac{N_{PS}(x_1, \dots, x_n)}{N} = S_{PS}(x_1, \dots, x_n) = P(x_1, \dots, x_n) \quad (14.4)$$

Например, рассмотрим событие, полученное ранее:  $[true, false, true, true]$ . Вероятность формирования выборки для этого события такова:

$$S_{PS}(true, false, true, true) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324$$

Поэтому следует полагать, что в пределе, при очень больших значениях  $N$ , около 32,4% выборок будут относиться к этому событию.

В последующем изложении мы будем использовать знак приближенного равенства ( $\approx$ ) для обозначения соотношения, имеющего именно этот смысл, — что оцениваемая вероятность становится точной при больших пределах количества выборок. Такая оценка называется **согласованной**. Например, может быть получена согласованная оценка вероятности любого частично заданного события  $x_1, \dots, x_m$ , где  $m \leq n$ , следующим образом:

$$P(x_1, \dots, x_m) \approx N_{PS}(x_1, \dots, x_m) / N \quad (14.5)$$

Это означает, что вероятность события можно оценить с помощью деления количества выборок частично заданного события на количество всех событий, получен-

ных в процессе формирования выборок. Например, если на основании сети с описанием опрыскивателя (см. рис. 14.9, а) сформирована 1000 выборок и для 511 из них справедливо выражение  $\text{Rain}=\text{true}$ , то оценка вероятности дождя, которая записывается как  $\hat{P}(\text{Rain}=\text{true})$ , равна 0,511.

### Формирование выборок с исключением в байесовских сетях

❖ **Формирование выборок с исключением** представляет собой общий метод получения выборок на основании распределения, для которого трудно получить выборки, если дано распределение, позволяющее легко сформировать выборки. В своей простейшей форме этот метод может использоваться для вычисления условных вероятностей, т.е. для определения вероятностей  $P(X|e)$ . Алгоритм Rejection-Sampling приведен в листинге 14.4. Вначале он формирует выборки из априорного распределения, определяемого сетью, а затем исключает все те выборки, которые не соответствуют свидетельству. Наконец, формируется оценка  $\hat{P}(X=x|e)$  путем подсчета того, насколько часто событие  $X=x$  встречалось в оставшихся выборках.

**Листинг 14.4. Алгоритм формирования выборок с исключением для получения ответов на запросы, если даны свидетельства в байесовской сети**

---

```

function Rejection-Sampling( $X, e, bn, N$ ) returns оценка значения  $P(X|e)$ 
  inputs:  $X$ , переменная запроса
            $e$ , свидетельство, определяемое как некоторое событие
            $bn$ , байесовская сеть
            $N$ , общее количество выборок, которые должны
           быть сформированы
  local variables:  $\mathbf{N}$ , вектор результатов подсчетов по  $X$ ,
                     первоначально равный нулю

  for  $j = 1$  to  $N$  do
     $\mathbf{x} \leftarrow \text{Prior-Sample}(bn)$ 
    if выборка  $\mathbf{x}$  согласуется со свидетельством  $e$  then
       $\mathbf{N}[x] \leftarrow \mathbf{N}[x]+1$ , где  $x$  представляет собой значение
      переменной  $X$  в множестве  $\mathbf{x}$ 
  return Normalize( $\mathbf{N}[X]$ )

```

---

Допустим, что  $\hat{P}(X|e)$  — оцениваемое распределение, которое возвращено алгоритмом. Из определения алгоритма получаем следующее:

$$\hat{P}(X|e) = \alpha \mathbf{N}_{PS}(X, e) = \frac{\mathbf{N}_{PS}(X, e)}{N_{PS}(e)}$$

С помощью уравнения 14.5 это соотношение может быть преобразовано в следующее:

$$\hat{P}(X|e) \approx \frac{P(X, e)}{P(e)} = P(X|e)$$

Таким образом, алгоритм формирования выборок с исключением вырабатывает согласованную оценку истинной вероятности.

Продолжая пример, приведенный на рис. 14.9, а, предположим, что необходимо оценить вероятность  $P(\text{Rain}|\text{Sprinkler}=\text{true})$  с использованием 100 выборок. Допустим, что из 100 сформированных выборок 73 соответствуют событию

*Sprinkler=false* и исключаются, а для 27 имеет место *Sprinkler=true*; из 27 выборок в 8 наблюдается событие *Rain=true*, а в 19 — *Rain=false*. Поэтому получаем следующее:

$$\mathbf{P}(\text{Rain} | \text{Sprinkler}=\text{true}) \approx \text{Normalize}(<8, 19>) = <0.296, 0.704>$$

Правильным ответом является  $<0.3, 0.7>$ . В ходе дальнейшего накопления все большего количества выборок эта оценка сходится к правильному ответу. Среднеквадратичное отклонение ошибки в каждой оценке вероятности будет пропорционально  $1/\sqrt{n}$ , где  $n$  — количество выборок, используемых для получения оценки.

Самым большим недостатком алгоритма формирования выборок с исключением является то, что в нем приходится исключать слишком много выборок! Доля выборок, согласованных со свидетельством  $\mathbf{e}$ , уменьшается экспоненциально по мере увеличения количества переменных свидетельства, поэтому данная процедура становится неприменимой для решения сложных задач.

Обратите внимание на то, что процесс формирования выборок с исключением весьма напоминает процесс оценки условных вероятностей непосредственно по данным, полученным из реального мира. Например, чтобы оценить вероятность дождя после того, как накануне вечером наблюдался красный закат,  $\mathbf{P}(\text{Rain} | \text{RedSkyAtNight}=\text{true})$ , можно подсчитать, насколько часто наблюдался дождь после красного заката, игнорируя данные о тех вечерах, когда закат не был красным. (Поэтому в данном случае роль алгоритма формирования выборок играет сама природа.) Безусловно, для проведения таких наблюдений может потребоваться много времени, если закат бывает красным очень редко, и в этом состоит недостаток процедуры формирования выборок с исключением.

### Оценка веса с учетом правдоподобия

В методе **оценки веса с учетом правдоподобия** преодолен указанный недостаток метода формирования выборок с исключением благодаря тому, что в нем вырабатываются только события, согласованные со свидетельством  $\mathbf{e}$ . Начнем с описания того, как работает алгоритм, затем покажем, что он работает правильно, т.е. вырабатывает согласованные оценки вероятности.

В алгоритме *Likelihood-Weighting*, приведенном в листинге 14.5, значения для переменных свидетельства  $\mathbf{E}$  фиксируются и формируются выборки только для оставшихся переменных  $X$  и  $Y$ . Это позволяет гарантировать, что каждое выработанное событие будет согласованным со свидетельством. Но не все события являются равноправными. Поэтому перед подведением итогов подсчетов в распределении для переменной запроса каждое событие взвешивается с учетом правдоподобия того, что событие согласуется со свидетельством. Такое правдоподобие измеряется с помощью произведения условных вероятностей для каждой переменной свидетельства, если даны ее родительские переменные. Интуитивно ясно, что события, в которых фактическое свидетельство кажется маловероятным, должны получать меньший вес.

**Листинг 14.5. Алгоритм оценки веса с учетом правдоподобия, предназначенный для вероятностного вывода в байесовских сетях**

---

```

function Likelihood-Weighting( $X$ ,  $\mathbf{e}$ ,  $bn$ ,  $N$ ) returns оценка
    значения  $P(X|\mathbf{e})$ 
    inputs:  $X$ , переменная запроса
     $\mathbf{e}$ , свидетельство, определяемое как некоторое событие

```

```

 $bn$ , байесовская сеть
 $N$ , общее количество выборок, которые должны
быть сформированы
local variables:  $W$ , вектор взвешенных результатов подсчетов по  $X$ ,
первоначально равный нулю

for  $j = 1$  to  $N$  do
     $\mathbf{x}, w \leftarrow \text{Weighted-Sample}(bn)$ 
     $W[\mathbf{x}] \leftarrow W[\mathbf{x}] + w$ , где  $\mathbf{x}$  представляет собой значение переменной  $X$ 
    в множестве  $\mathbf{x}$ 
return  $\text{Normalize}(W[X])$ 

function  $\text{Weighted-Sample}(bn, e)$  returns событие  $\mathbf{x}$  и вес  $w$ 

     $\mathbf{x} \leftarrow$  событие с  $n$  элементами;  $w \leftarrow 1$ 
    for  $i = 1$  to  $n$  do
        if  $X_i$  имеет значение  $x_i$  in свидетельство  $e$ 
            then  $w \leftarrow w \times P(X_i=x_i | \text{parents}(X_i))$ 
        else  $x_i \leftarrow$  случайная выборка из  $P(X_i | \text{parents}(X_i))$ 
    return  $\mathbf{x}, w$ 

```

---

Применим этот алгоритм к сети, показанной на рис. 14.9, *a*, на примере получения ответа на запрос  $P(\text{Rain} | \text{Sprinkler}=\text{true}, \text{WetGrass}=\text{true})$ . Этот процесс происходит следующим образом: вначале вес  $w$  устанавливается равным 1,0, затем вырабатывается событие, как описано ниже.

1. Выборка из распределения  $P(\text{Cloudy}) = <0.5, 0.5>$ ; предположим, что она возвращает значение *true*.
2. *Sprinkler* — переменная доказательства со значением *true*. Поэтому мы устанавливаем:
 
$$w \leftarrow w \times P(\text{Sprinkler}=\text{true} | \text{Cloudy}=\text{true}) = 0.1$$
3. Выборка из распределения  $P(\text{Rain} | \text{Cloudy}=\text{true}) = <0.8, 0.2>$ ; предположим, что она возвращает значение *true*.
 
$$w \leftarrow w \times P(\text{Rain}=\text{true} | \text{Cloudy}=\text{true}) = 0.8$$
4. *WetGrass* — переменная доказательства со значением *true*. Поэтому мы устанавливаем:
 
$$w \leftarrow w \times P(\text{WetGrass}=\text{true} | \text{Sprinkler}=\text{true}, \text{Rain}=\text{true}) = 0.099$$

Теперь алгоритм *Weighted-Sample* возвращает событие  $[\text{true}, \text{true}, \text{true}, \text{true}]$  с весом 0,099, и данные об этом событии подытоживаются с учетом условия  $\text{Rain}=\text{true}$ . Этот вес является низким потому, что событие описывает пасмурный день, в который вероятность применения опрыскивателя является весьма небольшой.

Чтобы понять, как работает алгоритм оценки веса с учетом правдоподобия, начнем с изучения сформированного с помощью выборок распределения  $S_{\text{WS}}$  для функции *Weighted-Sample*. Напомним, что переменные свидетельства  $\mathbf{E}$  зафиксированы со значениями  $\mathbf{e}$ . Обозначим все прочие переменные как  $\mathbf{Z}$ , иными словами,  $\mathbf{Z} = \{X\} \cup \mathbf{Y}$ . В алгоритме выборки для каждой переменной из  $\mathbf{Z}$ , если даны ее родительские значения, формируются следующим образом:

$$S_{\text{WS}}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^1 P(z_i | \text{parents}(z_i)) \quad (14.6)$$

Обратите внимание на то, что в число переменных  $Parents(Z_i)$  могут входить и скрытые переменные, и переменные свидетельства. В отличие от априорного распределения  $P(\mathbf{z})$ , в распределении  $S_{ws}$  некоторое внимание уделено свидетельству: на значения сформированных выборок для каждой переменной  $Z_i$ , кроме других предков  $Z_i$ , оказывает влияние само свидетельство. С другой стороны, в распределении  $S_{ws}$  свидетельству уделяется меньше внимания, чем в истинном апостериорном распределении  $P(\mathbf{z} | \mathbf{e})$ , поскольку в значениях сформированных выборок для каждой переменной  $Z_i$  игнорируются свидетельства<sup>6</sup>, относящиеся к переменным, которые не являются предками  $Z_i$ .

Весовое значение правдоподобия  $w$  представляет собой разность между фактическим и желаемым распределениями вероятностей сформированных выборок. Такой вес для данной конкретной выборки  $\mathbf{x}$ , состоящий из значений  $\mathbf{z}$  и  $\mathbf{e}$ , является произведением значений правдоподобия каждой переменной свидетельства, если даны ее родительские переменные (причем некоторые или все эти переменные могут находиться среди переменных  $Z_i$ ):

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^m P(e_i | parents(E_i)) \quad (14.7)$$

Умножая уравнения 14.6 и 14.7, можно определить, что соотношение для взвешенной вероятности выборки имеет следующую особенно удобную форму, поскольку эти два произведения охватывают все переменные, заданные в сети:

$$\begin{aligned} S_{ws}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) &= \prod_{i=1}^1 P(z_i | parents(Z_i)) \prod_{i=1}^m P(e_i | parents(E_i)) \\ &= P(\mathbf{z}, \mathbf{e}) \end{aligned} \quad (14.8)$$

Это позволяет использовать для вычисления совместной вероятности уравнение 14.1.

Теперь можно легко показать, что оценки весов, полученные с учетом правдоподобия, являются согласованными. Для любого конкретного значения  $x$  переменной  $X$  оценка апостериорной вероятности может быть рассчитана следующим образом:

$$\begin{aligned} P(x | \mathbf{e}) &= \alpha \sum_{\mathbf{y}} N_{ws}(x, \mathbf{y}, \mathbf{e}) w(x, \mathbf{y}, \mathbf{e}) \quad \text{из алгоритма Likelihood-Weighting} \\ &\approx \alpha' \sum_{\mathbf{y}} S_{ws}(x, \mathbf{y}, \mathbf{e}) w(x, \mathbf{y}, \mathbf{e}) \quad \text{для больших } N \end{aligned}$$

<sup>6</sup> Желательно было бы использовать распределение вероятностей сформированных выборок, равное истинному апостериорному распределению  $P(\mathbf{z} | \mathbf{e})$ , чтобы в расчет принимались все свидетельства. Но такой подход не может быть осуществлен эффективно, поскольку если бы существовала такая возможность, то мы могли бы оценивать желаемую вероятность с любой заданной точностью, применяя полиномиальное количество выборок. Тем не менее можно показать, что существование такой полиномиальной схемы аппроксимации невозможно.

$$\begin{aligned}
 &= \alpha' \sum_{\mathbf{y}} P(x, \mathbf{y}, \mathbf{e}) \\
 &= \alpha' P(x, \mathbf{e}) = P(x | \mathbf{e})
 \end{aligned}
 \quad \text{согласно уравнению 14.8}$$

Поэтому алгоритм оценки веса с учетом правдоподобия возвращает согласованные оценки.

Поскольку в алгоритме оценки веса с учетом правдоподобия используются все сформированные выборки, он может оказаться гораздо более эффективным по сравнению с алгоритмом формирования выборок с исключением. Тем не менее его недостатком является снижение производительности по мере увеличения количества переменных свидетельства. Это связано с тем, что в таком случае большинство выборок имеет очень низкие веса, поэтому в составе взвешенной оценки доминирует крошечная доля выборок, которые согласуются со свидетельством с правдоподобием, большим бесконечно малого. Эта проблема усугубляется, если переменные свидетельства расположены на последних местах в упорядочении переменных, поскольку в таком случае процесс формирования выборок представляет собой процесс моделирования, имеющий мало сходства с реальностью и имитируемый с помощью свидетельства.

### Вероятностный вывод по методу моделирования цепи Маркова

В этом разделе описан алгоритм **Монте-Карло с применением цепи Маркова** (Markov chain Monte Carlo — MCMC), предназначенный для вероятностного вывода в байесовских сетях. Вначале опишем, какие действия выполняются в этом алгоритме, затем объясним, благодаря чему он работает и почему имеет такое сложное название.

#### Алгоритм MCMC

В отличие от других двух алгоритмов формирования выборок, которые вырабатывают каждое событие с нуля, в алгоритме MCMC каждое событие вырабатывается путем внесения случайного изменения в предыдущее событие. Поэтому сеть целесообразно рассматривать как находящуюся в конкретном текущем состоянии, заданном с помощью присваивания значения каждой переменной. Переход в следующее состояние осуществляется путем случайного формирования выборки значения для одной из переменных  $X_i$ , отличных от переменных свидетельства, причем это значение обусловлено текущими значениями переменных в марковском покрытии переменной  $X_i$ . (Напомним, что, как было сказано на с. 669, марковское покрытие переменной состоит из ее родительских переменных, дочерних переменных и родительских переменных дочерних переменных.) Поэтому в алгоритме MCMC предусмотрено случайное блуждание по пространству состояний (пространству возможных полных присваиваний), в котором каждый раз изменяется значение одной переменной, но значения переменных свидетельства остаются зафиксированными.

Рассмотрим запрос  $P(Rain | Sprinkler=true, WetGrass=true)$ , примененный к сети, которая показана на рис. 14.9, а. Для переменных свидетельства *Sprinkler* и *WetGrass* зафиксированы их наблюдаемые значения, а скрытые переменные *Cloudy* и *Rain* инициализированы случайным образом; допустим, что им присвоены значения *true* и *false* соответственно. Таким образом, начальным

состоянием является  $[true, true, false, true]$ . После этого повторно выполняются описанные ниже этапы.

1. Формируется выборка для переменной *Cloudy* с учетом текущих значений переменных марковского покрытия: в данном случае выборка берется из  $\mathbf{P}(\text{Cloudy} \mid \text{Sprinkler}=\text{true}, \text{Rain}=\text{false})$ . (Вскоре мы покажем, как рассчитать это распределение.) Предположим, что результатом является *Cloudy=false*. В таком случае новым текущим состоянием становится  $[false, true, false, true]$ .
2. Формируется выборка для переменной *Rain* с учетом текущих значений переменных марковского покрытия: в данном случае выборка берется из  $\mathbf{P}(\text{Rain} \mid \text{Cloudy}=\text{false}, \text{Sprinkler}=\text{true}, \text{WetGrass}=\text{true})$ . Предположим, что эта операция приводит к получению *Rain=true*. Новым текущим состоянием становится  $[false, true, true, true]$ .

Каждое состояние в пространстве состояний, посещенное в ходе этого процесса, представляет собой выборку, которая вносит свой вклад в оценку вероятности переменной запроса *Rain*. Если в данном процессе посещаются 20 состояний, в которых переменная *Rain* принимает истинное значение, и 60 состояний, в которых переменная *Rain* становится ложной, то ответом на запрос становится  $\text{Normalize}(<20, 60>) = <0.25, 0.75>$ . Полный алгоритм показан в листинге 14.6.

#### Листинг 14.6. Алгоритм МСМС приближенного вероятностного вывода в байесовских сетях

---

```

function MCMC-Ask(X, e, bn, N) returns оценка значения  $P(X \mid e)$ 
  local variables: N[X], вектор результатов подсчетов по X,
    первоначально равный нулю
    Z, переменные в сети bn, отличные от переменных
    свидетельства
    x, текущее состояние сети, первоначально
    скопированное из e

  инициализировать x случайными значениями переменных из Z
  for j = 1 to N do
    for each Zi in Z do
      выполнить выборку значения переменной Zi в векторе x
      из  $\mathbf{P}(Z_i \mid mb(Z_i))$ , если даны значения MB(Zi) в векторе x
      N[x] ← N[x]+1, где x представляет собой значение переменной X
      в множестве x
  return  $\text{Normalize}(\mathbf{N}[X])$ 

```

---

#### Обоснование правильности работы алгоритма МСМС

В этом разделе будет показано, что алгоритм МСМС возвращает согласованные оценки для апостериорных вероятностей. Материал, изложенный в данном разделе, является весьма формальным, но основное утверждение в нем несложно: *процесс формирования выборок переходит в “динамическое равновесие”, в котором в конечном итоге доля времени, проведенного в каждом состоянии, точно пропорциональна апостериорной вероятности этого состояния*. Такое замечательное свойство является следствием конкретной *переходной вероятности*, с которой данный процесс переходит из од-

ногого состояния в другое и которая определена условным распределением, заданным с учетом марковского покрытия переменной, для которой формируется выборка.

Предположим, что  $q(\mathbf{x} \rightarrow \mathbf{x}')$  — вероятность того, что в этом процессе произойдет переход из состояния  $\mathbf{x}$  в состояние  $\mathbf{x}'$ . Эта переходная вероятность определяет информационную структуру, заданную на пространстве состояний, которая называется **цепью Маркова**. (Цепи Маркова играют также важную роль в главах 15 и 17.) Теперь предположим, что мы развернули цепь Маркова на  $t$  этапов, и допустим, что  $\pi_t(\mathbf{x})$  — вероятность того, что система находится в состоянии  $\mathbf{x}$  во время  $t$ . Аналогичным образом, допустим, что  $\pi_{t+1}(\mathbf{x}')$  — вероятность пребывания системы в состоянии  $\mathbf{x}'$  во время  $t+1$ . Если дано значение  $\pi_t(\mathbf{x})$ , то значение  $\pi_{t+1}(\mathbf{x}')$  можно рассчитать путем суммирования по всем состояниям, в которых система может находиться во время  $t$ , вероятностей пребывания в этом состоянии, умноженных на вероятности осуществления перехода в состояние  $\mathbf{x}'$ , следующим образом:

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}')$$

Цепь называется достигшей своего **стационарного распределения** (stationary distribution), если  $\pi_t = \pi_{t+1}$ . Назовем это стационарное распределение  $\pi$ ; итак, его определяющим уравнением является следующее:

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \text{ для всех } \mathbf{x}' \quad (14.9)$$

При некоторых стандартных допущениях<sup>7</sup>, касающихся распределения вероятностей перехода  $q$ , существует одно и только одно распределение  $\pi$ , удовлетворяющее этому уравнению при каждом конкретном значении  $q$ .

Уравнение 14.9 можно трактовать как утверждение, что в установившемся режиме ожидаемый “отток” из каждого состояния (т.е. его текущее “население”) равен ожидаемому “притоку” из всех других состояний. Один из очевидных способов удовлетворения этого отношения состоит в достижении того, чтобы ожидаемый поток между любыми парами состояний был одинаковым в обоих направлениях. В этом состоит свойство **детализированного равновесия**, которое показано ниже.

$$\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \text{ для всех } \mathbf{x}, \mathbf{x}' \quad (14.10)$$

Можно показать, что из этого свойства детализированного равновесия можно вывести свойство стационарности, получив суммы по  $\mathbf{x}$  в уравнении 14.10. Получаем следующее соотношение:

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

где возможен последний этап преобразования, поскольку гарантировано выполнение перехода из состояния  $\mathbf{x}'$ .

<sup>7</sup> Цепь Маркова, определяемая значением  $q$ , должна быть **эргодичной**; это означает, что каждое состояние должно быть достижимо из любого другого состояния и не должно быть строго периодических циклов.

Теперь мы покажем, что вероятность перехода  $q(\mathbf{x} \rightarrow \mathbf{x}')$ , определяемая на этапе формирования выборки в алгоритме MCMC-Ask, удовлетворяет уравнению детализированного равновесия со стационарным распределением, равным  $P(\mathbf{x} | \mathbf{e})$  (истинному апостериорному распределению по скрытым переменным). Мы проведем это доказательство в два этапа. Вначале определим цепь Маркова, в которой формирование выборки по каждой переменной обусловлено текущими значениями всех прочих переменных, и покажем, что это условие соответствует свойству детализированного равновесия. Затем мы просто констатируем, что для байесовских сетей формирование такой условной выборки эквивалентно условному формированию выборки по марковскому покрытию переменной (см. с. 669).

Допустим, что  $X_i$  — переменная, для которой должна быть сформирована выборка, и предположим, что  $\tilde{\mathbf{x}}_i$  — все скрытые переменные, отличные от  $X_i$ . Их значениями в текущем состоянии являются  $x_i$  и  $\tilde{\mathbf{x}}_i$ . Если будет выполнена выборка нового значения  $x_i'$  для переменной  $X_i$ , обусловленного всеми прочими переменными, включая переменные свидетельства, то будет получено следующее:

$$q(\mathbf{x} \rightarrow \mathbf{x}') = q((x_i, \tilde{\mathbf{x}}_i) \rightarrow (x_i', \tilde{\mathbf{x}}_i)) = P(x_i' | \tilde{\mathbf{x}}_i, \mathbf{e})$$

Это выражение для переходной вероятности называется **формирователем выборок Гиббса** (Gibbs sampler) и служит основой особенно удобной формы алгоритма MCMC. Теперь мы покажем, что формирователь выборок Гиббса находится в детализированном равновесии с истинной апостериорной вероятностью:

$$\begin{aligned} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') &= P(\mathbf{x} | \mathbf{e}) P(x_i' | \tilde{\mathbf{x}}_i, \mathbf{e}) = P(x_i, \tilde{\mathbf{x}}_i | \mathbf{e}) P(x_i' | \tilde{\mathbf{x}}_i, \mathbf{e}) \\ &= P(x_i | \tilde{\mathbf{x}}_i, \mathbf{e}) P(\tilde{\mathbf{x}}_i | \mathbf{e}) P(x_i' | \tilde{\mathbf{x}}_i, \mathbf{e}) \quad (\text{использование цепного правила} \\ &\quad \text{после первого терма}) \\ &= P(x_i | \tilde{\mathbf{x}}_i, \mathbf{e}) P(x_i', \tilde{\mathbf{x}}_i | \mathbf{e}) \quad (\text{использование цепного правила для} \\ &\quad \text{перехода в обратном направлении}) \\ &= \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \end{aligned}$$

Как указано на с. 669, переменная независима от всех других переменных, если дано ее марковское покрытие, поэтому имеет место следующее соотношение:

$$P(x_i' | \tilde{\mathbf{x}}_i, \mathbf{e}) = P(x_i' | mb(X_i))$$

где  $mb(X_i)$  обозначает значения переменных в марковском покрытии  $X_i$ ,  $MB(X_i)$ . Как показано в упр. 14.10, вероятность переменной, если дано ее марковское покрытие, пропорциональна вероятности переменной, если даны ее родительские переменные, умноженной на вероятность каждой дочерней переменной, если даны ее соответствующие родительские переменные:

$$P(x_i' | mb(X_i)) = \alpha P(x_i' | parents(X_i)) \times \prod_{Y_j \in Children(X_i)} P(Y_j | parents(Y_j)) \quad (14.11)$$

Поэтому для изменения значения каждой переменной  $X_i$  необходимо выполнить такое количество операций умножения, которое равно количеству дочерних переменных переменной  $X_i$ .

В этом разделе рассматривался только один простой вариант алгоритма MCMC — вариант, основанный на использовании формирователя выборок Гиббса.

В своей наиболее общей форме алгоритм MCMC представляет собой мощный метод вычислений с помощью вероятностных моделей, поэтому было разработано много вариантов этого алгоритма, включая алгоритм эмуляции отжига (см. главу 4), алгоритмы стохастической выполнимости (см. главу 7) и формирователь выборок Метрополис–Гастингса, рассматриваемый в главе 15.

## 14.6. РАСПРОСТРАНЕНИЕ ВЕРОЯТНОСТНЫХ МЕТОДОВ НА ПРЕДСТАВЛЕНИЯ В ЛОГИКЕ ПЕРВОГО ПОРЯДКА

В главе 8 описывалось, какими преимуществами с точки зрения возможностей представления обладает логика первого порядка по сравнению с пропозициональной логикой. В логике первого порядка учитывается существование объектов и отношений между ними, и она позволяет выражать факты о некоторых или обо всех объектах в проблемной области. Это часто приводит к созданию представлений, намного более кратких, чем эквивалентные пропозициональные описания. Теперь отметим, что байесовские сети по существу являются пропозициональными: множество переменных в них является фиксированным и конечным, а каждая переменная имеет постоянную область определения, состоящую из возможных значений. В связи с этим применимость байесовских сетей становится ограниченной. *Если бы можно было найти способ применения теории вероятностей в сочетании с выразительными возможностями представлений в логике первого порядка, то можно было бы рассчитывать на существенное расширение спектра задач, решаемых таким образом.*

Основная идея, которая может привести к достижению этой цели, состоит в следующем: в пропозициональном контексте байесовская сеть задает распределения вероятностей по атомарным событиям, каждое из которых определяет значение для каждой переменной в сети. Таким образом, атомарное событие является **моделью**, или одним из **возможных миров**, в терминологии пропозициональной логики. А в контексте логики первого порядка модель (с ее интерпретацией) задает область определения объектов, отношения, которые имеют место между этими объектами, и отображения из констант и предикатов базы знаний в объекты и отношения модели. Поэтому *вероятностная база знаний первого порядка должна задавать вероятности для всех возможных моделей первого порядка*. Допустим, что  $\mu(M)$  — вероятность, присвоенная модели  $M$  с помощью базы знаний. Для каждого высказывания в логике первого порядка  $\phi$  вероятность  $P(\phi)$  задается обычным образом, путем суммирования по всем возможным мирам, где высказывание  $\phi$  является истинным:

$$P(\phi) = \sum_{M:\phi \text{ является истинным в } M} \mu(M) \quad (14.12)$$

До сих пор все, казалось бы, шло хорошо. Тем не менее есть одна проблема: множество моделей первого порядка является бесконечным. Это означает, что, во-первых, суммирование может быть неосуществимым, и, во-вторых, задача определения полного, согласованного распределения на бесконечном множестве миров может оказаться очень трудной.

Поэтому мы должны умерить наши амбиции, по крайней мере, на время. В частности, определим ограниченный язык, для которого интересующими становятся только модели, количество которых является конечным. Для этого можно использовать несколько способов. В этом разделе будут описаны **реляционные вероятностные модели**, или RPM (Relational Probability Model), идея которых заимствована из семантических сетей (см. главу 10) и из объектно-реляционных баз данных. Другие подходы обсуждаются в библиографических и исторических заметках в конце настоящей главы.

Модели RPM допускают использование константных символов, которыми именуются объекты. Например, допустим, что *ProfSmith* — имя профессора, а *Jones* — имя студента. Каждый объект представляет собой экземпляр некоторого класса; например, *ProfSmith* относится к классу *Professor* (Профессор), *Jones* — к классу *Student* (Студент). Предполагается, что класс каждого константного символа известен.

Применяемые нами функциональные символы будут подразделяться на два типа. Функции первого типа, **простые функции**, отображают объект не на другой структурированный объект, а на некоторое значение из фиксированной области значений, полностью аналогично случайной переменной. Например, значениями функций *Intelligence(Jones)* (Интеллект Джонса) и *Funding(ProfSmith)* (Финансирование профессора Смита) могут быть *hi* (высокий) или *lo* (низкий), а значениями функций *Success(Jones)* (Успех Джонса) и *Fame(ProfSmith)* (Известность профессора Смита) могут быть *true* (истинный) или *false* (ложный). Функциональные символы не должны применяться к таким значениям, как *true* и *false*, поэтому вложение простых функций невозможно. Это позволяет исключить один из источников бесконечных ветвлений. Значение простой функции, применяемое к конкретному объекту, может быть наблюдаемым или неизвестным; для нашего представления эти значения будут служить значениями основных случайных переменных<sup>8</sup>.

Допускается также использование **сложных функций**, которые отображают объекты на другие объекты. Например, значением функции *Advisor(Jones)* (Консультант Джонса) может быть *ProfSmith*. Каждая сложная функция имеет заданную область определения и диапазон значений, которые являются классами. Например, областью определения функции *Advisor* является *Student*, а диапазоном значений — *Professor*. Функции применяются только к объектам соответствующего им класса; например, значение консультанта *Advisor* для *ProfSmith* не определено. Сложные функции могут вкладываться, например значением выражения *DeptHead(Advisor(Jones))* может быть *ProfMoore*. На данный момент предполагается, что для всех константных символов значения всех сложных функций известны. Поскольку база знаний является конечной, из этого следует, что каждая цепочка применений сложных функций приводит к получению одного объекта из конечного множества объектов<sup>9</sup>.

<sup>8</sup> Эти значения играют роль, во многом аналогичную роли базовых атомарных высказываний, формируемых в процессе пропозиционализации, который описан в разделе 9.1.

<sup>9</sup> Это ограничение означает, что не могут использоваться такие сложные функции, как *Father* и *Mother*, приводящие к созданию потенциально бесконечных цепочек, которые могли бы оканчиваться формированием неизвестного объекта. Указанное ограничение будет пересмотрено позднее.

Последним необходимым нам элементом является вероятностная информация. Для каждой простой функции задается множество родительских объектов, так же, как и в байесовских сетях. Родительскими объектами могут быть другие простые функции от того же объекта; например, финансирование (*Funding*) некоторого профессора (*Professor*) может зависеть от его известности (*Fame*). Кроме того, родительские объекты могут быть простыми функциями от связанных объектов, например, успех (*Success*) студента может зависеть от его интеллекта (*Intelligence*) и от известности (*Fame*) консультанта этого студента. По сути подобные высказывания представляют собой утверждения о родительских объектах всех объектов в классе с квантором всеобщности. Таким образом, можно записать следующее:

$$\forall x \ x \in \text{Student} \Rightarrow \\ \text{Parents}(\text{Success}(x)) = \{\text{Intelligence}(x), \text{Fame}(\text{Advisor}(x))\}$$

(При мене формальном подходе можно нарисовать диаграммы, подобные приведенной на рис. 14.10, а.) Теперь определим распределение условных вероятностей для дочернего объекта, если даны значения его родительских объектов. Например, можно сформулировать следующее утверждение:

$$\forall x \ x \in \text{Student} \Rightarrow \\ P(\text{Success}(x) = \text{true} | \text{Intelligence}(x) = \text{hi}, \text{Fame}(\text{Advisor}(x)) = \text{true}) = 0.95$$

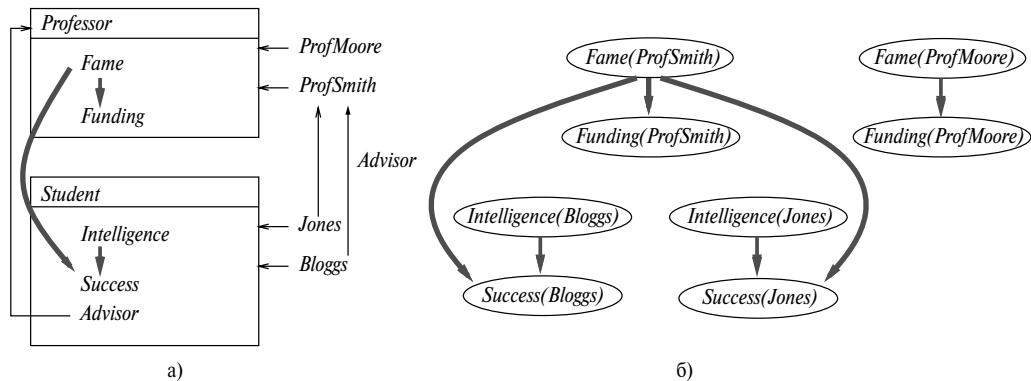


Рис. 14.10. Примеры диаграмм: модель RPM с описанием двух классов — Professor и Student. Имеются два профессора и два студента, причем ProfSmith является консультантом обоих студентов (а); байесовская сеть, эквивалентная модели RPM, приведенной на рис. 14.10, а (б)

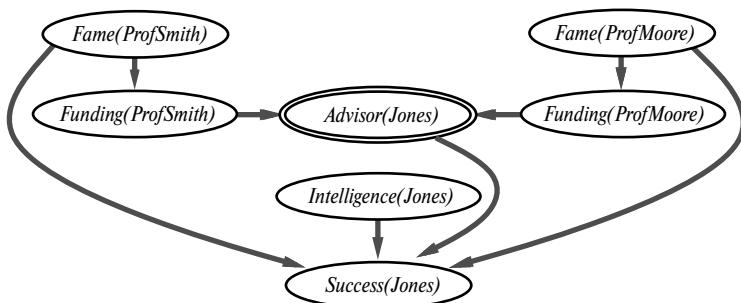
Как и в семантических сетях, распределение условных вероятностей можно закрепить за самим классом так, чтобы его экземпляры **наследовали** зависимости и условные вероятности от этого класса.

В семантике для языка RPM подразумевается, что каждый константный символ ссылается на отдельный объект; в этом состоит **предположение об уникальности имен**, описанное в главе 10. С учетом этого определения и ограничений, перечисленных выше, можно показать, что каждая модель RPM вырабатывает фиксированное конечное множество случайных переменных, причем каждая из них является результатом применения простой функции к константному символу. Таким образом, при условии, что родительско-дочерние зависимости являются ациклическими, можно составить эквивалентную байесовскую сеть. Это означает, что модель RPM и

байесовская сеть задают идентичные вероятности для каждого возможного мира. На рис. 14.10, б показана байесовская сеть, соответствующая модели RPM, приведенной на рис. 14.10, а. Обратите внимание на то, что связи *Advisor*, имеющиеся в модели RPM, в байесовской сети отсутствуют. Это связано с тем, что такие связи являются фиксированными и известными. Однако они присутствуют в топологии сети неявно, например, объект *Success(Jones)* имеет в качестве родительского объект *Fame(ProfSmith)*, поскольку значением функции *Advisor(Jones)* является *ProfSmith*. Вообще говоря, отношения, которые имеют место между объектами, определяют характер зависимостей между свойствами этих объектов.

Для повышения выразительной мощи моделей RPM применяется несколько способов. В частности, могут быть разрешены **рекурсивные зависимости** между переменными, позволяющие представить несколько типов зависимостей, которые ссылаются сами на себя. Например, предположим, что склонность к питанию всухомятку вызвана фактором *McGene*. В таком случае для любого  $x$  истинность выражения *McGene(x)* зависит от *McGene(Father(x))* и *McGene(Mother(x))*, которые, в свою очередь, зависят от *McGene(Father(Father(x)))*, *McGene(Mother(Father(x)))*, и т.д. Даже несмотря на то, что такие базы знаний соответствуют байесовским сетям с бесконечно большим количеством случайных переменных, иногда решения могут быть получены на основе уравнений с неподвижной точкой (fixed-point equation). Например, равновесное распределение вероятностей для генетического фактора *McGene* можно рассчитать на основе условной вероятности наследования этого фактора. Еще одно очень важное семейство рекурсивных баз знаний состоит из **временных вероятностных моделей**, которые описаны в главе 15. В этих моделях свойства рассматриваемого состояния во время  $t$  зависят от свойств этого состояния во время  $t-1$  и т.д.

Модели RPM могут быть также расширены, чтобы в них можно было представить **реляционную неопределенность**, т.е. неопределенность, касающуюся значений сложных функций. Например, в базе знаний может отсутствовать информация о том, кто является консультантом Джонса, *Advisor(Jones)*. В таком случае *Advisor(Jones)* становится случайной переменной с возможными значениями *ProfSmith* и *ProfMoore*. Соответствующая сеть показана на рис. 14.11.



*Рис. 14.11. Часть байесовской сети, соответствующая одной из моделей RPM, в которой значение *Advisor(Jones)* неизвестно, но им может быть либо *ProfSmith*, либо *ProfMoore*. Выбор студентом консультанта зависит от того, какой объем финансирования имеет каждый профессор. Обратите внимание на то, что теперь успех Джонса, *Success(Jones)*, зависит от известности, *Fame*, обоих профессоров, хотя тот факт, какой из этих профессоров окажет влияние на успех Джонса, фактически зависит от значения переменной *Advisor(Jones)**

Модели RPM позволяют также представить такое свойство, как ~~и~~ **неопределенность идентичности** (*identity uncertainty*); например, в базе знаний может отсутствовать информация о том, что *Mary* и *ProfSmith* — одно и то же лицо. При наличии свойства неопределенности идентичности количество объектов и высказываний в разных возможных мирах может изменяться. В мире, где *Mary* и *ProfSmith* — одно и то же лицо, количество объектов меньше, чем в том мире, где они считаются разными людьми. В результате этого процесс вероятностного вывода усложняется, но основные принципы, установленные в уравнении 14.12, все еще соблюдаются: вероятность любого высказывания вполне определена и может быть рассчитана. Свойство неопределенности идентичности является особенно важным для роботов и встроенных систем датчиков, которые должны следить за многими объектами. Мы вернемся к описанию этой проблемы в главе 15.

Теперь рассмотрим проблему вероятностного вывода. Очевидно, что такой вывод может осуществляться в эквивалентной байесовской сети при условии, что язык RPM будет ограничен таким образом, чтобы эквивалентная сеть была конечной и имела фиксированную структуру. Такой подход аналогичен способу, с помощью которого логический вывод в логике первого порядка может быть выполнен посредством логического вывода в пропозициональной логике по эквивалентной пропозициональной базе знаний (см. раздел 9.1). Но, как и в случае логического вывода, эквивалентная сеть может оказаться слишком большой для того, чтобы существовала возможность ее построить, не говоря уже о том, чтобы выполнять в ней вычисления. Кроме того, при этом возникает проблема плотных взаимосвязей (см. упр. 14.12). Поэтому для вероятностного вывода на основе модели RPM очень полезными являются приближенные алгоритмы, такие как MCMC (см. раздел 14.5).

В ходе применения алгоритма MCMC к байесовской сети, которая эквивалентна простой базе знаний RPM без реляционной неопределенности или неопределенности идентичности, алгоритм формирует выборки из пространства возможных миров, определяемых значениями простых функций, заданных на объектах. Можно легко показать, что такой подход может быть также дополнен с учетом реляционной неопределенности или неопределенности идентичности. В таком случае переход между возможными мирами способен привести к изменению значения простой функции или сложной функции и поэтому вызвать изменение структуры зависимостей. Кроме того, переходы могут стать причиной изменения отношений идентичности между константными символами. Поэтому создается впечатление, что алгоритм MCMC представляет собой изящный способ осуществления вероятностного вывода на основе весьма выразительных вероятностных баз знаний в логике первого порядка.

Исследования в этой области все еще находятся на ранних этапах, но уже стало очевидно, что вероятностные рассуждения в логике первого порядка позволяют достичь колossalного повышения эффективности применения систем искусственного интеллекта для обработки неопределенной информации. К числу потенциальных приложений относятся машинное зрение, обработка естественного языка, выборка информации и оценка ситуаций. Во всех этих областях множество объектов (и поэтому множество случайных переменных) заранее не известно, поэтому “чисто пропозициональные” методы, такие как байесовские сети, не способны полностью представить ситуацию. Была предпринята попытка расширить возможности таких сетей с помощью метода поиска в пространстве моделей, но подходы, основанные на использовании моделей RPM, обеспечивают формирование рассуждений об указанных видах неопределенности в рамках одной модели.

## 14.7. ДРУГИЕ ПОДХОДЫ К ФОРМИРОВАНИЮ РАССУЖДЕНИЙ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ

Применение вероятности в качестве модели для неопределенности уже очень давно стало доминирующим методом исследований не только в искусственном интеллекте, но и в других науках (например, в физике, генетике и экономике). В 1819 году Пьер Лаплас заявил: “Теория вероятностей есть не что иное, как здравый смысл, который удалось свести к вычислениям”, а в 1850 году Джеймс Максвелл сказал, что “истинной логикой для нашего мира является исчисление вероятностей, в котором учитывается та величина вероятности, какой она является или должна быть по оценке рассудительного человека”.

Поскольку существует такая долгая традиция в использовании вероятностных методов, может на первый взгляд показаться удивительным, что в искусственном интеллекте рассматривалось много альтернативных подходов, не предусматривающих применение вероятностей. В ранних экспертных системах 1970-х годов игнорировалась неопределенность и использовались строго логические рассуждения, но вскоре стало очевидно, что такой подход является практически не применимым для большинства проблемных областей реального мира. Поэтому в следующем поколении экспертных систем (особенно в проблемных областях медицины) использовались вероятностные методы. Первые результаты этого оказались многообещающими, но сами эти системы не могли быть развернуты в больших масштабах из-за экспоненциального количества вероятностей, которые требовались в полном совместном распределении (тогда еще не были известны эффективные алгоритмы для байесовских сетей). В результате этого вероятностные подходы в период 1975–1988 гг. перестали привлекать интерес исследователей, и по многим причинам, описанным ниже, был опробован целый ряд альтернативных подходов.

- Один из широко распространенных взглядов состоит в том, что теория вероятностей по сути является числовой, тогда как рассуждения человека, в которых выражаются его оценки, в большей степени имеют “качественный” характер. Безусловно, люди сознательно не выполняют числовые расчеты со степенями уверенности (к тому же люди не осознают те ситуации, в которых они выполняют унификацию, но, по-видимому, обладают способностью формировать логические рассуждения определенного вида). Тем не менее может оказаться, что числовые оценки степеней уверенности определенного рода кодируются непосредственно в виде интенсивностей соединений и параметров активизации нейронов человеческого мозга. Если дело обстоит действительно так, то не стоит удивляться тому, что человеку так трудно сознательно оценить эти интенсивности. Следует также отметить, что механизмы качественного рассуждения могут быть сформированы непосредственно на основе теории вероятностей, поэтому доводы против использования людьми вероятностей, поскольку они “не мыслят с помощью чисел”, можно легко опровергнуть. Однако некоторые схемы качественных рассуждений сами по себе оказались весьма привлекательными. Одним из наиболее хорошо исследованных подходов является **формирование рассуждений по умолчанию**, в котором логические заключения рассматриваются не как “являющиеся предметом уверенности в их истинности до определенной степени”, а как “являющиеся предметом уверенности в их

истинности до тех пор, пока не будет обнаружена более весомая причина для того, чтобы предметом уверенности в их истинности не стало нечто иное”. Методы формирования рассуждений по умолчанию рассматривались в главе 10.

- Были также предприняты попытки учета неопределенности с помощью подходов **на основе правил**. Приверженцы таких подходов надеялись развить успех логических систем на основе правил, но, как было признано в дальнейшем, в каждом правиле, в котором учитывалась неопределенность, в них приходилось добавлять своего рода “коэффициент подгонки под факты” (fudge factor). Эти методы разрабатывались в середине 1970-х годов и составили основу большого количества экспертных систем в медицине и других областях.
- Еще одна область, которая до сих пор не рассматривалась в этой книге, относится к вопросу о **незнании**, которое следует отличать от неопределенности. Рассмотрим задачу с подбрасыванием монеты. Если мы знаем, что монета подлинная, то вполне резонно можем считать, что вероятность выпадения орла равна 0,5. Если же известно, что монета поддельная, но не известно, есть ли на ней два орла или две решки, то по-прежнему единственным разумным решением является принятие гипотезы о вероятности 0,5. Очевидно, что оба эти случая не одинаковы, но теория вероятностей, по-видимому, не позволяет провести между ними различие. В **теории Демпстера-Шефера** для представления знаний агента о вероятности высказывания используются оценки степени уверенности **с интервальными значениями**. В литературе обсуждались также другие методы, в которых используются вероятности второго порядка.
- Теория вероятностей вносит такой же онтологический вклад в познание действительности, как и логика: в ней утверждается, что в рассматриваемом мире события являются истинными или ложными, даже если агент не может знать со всей определенностью, каковыми являются эти события. Исследователи в области **нечеткой логики** предложили онтологию, которая допускает **неосведомленность** (vagueness) — незнание о том, “до какой степени” является истинным некоторое событие. Как будет показано ниже, фактически проблемы неосведомленности и неопределенности являются ортогональными.

В следующих трех подразделах некоторые из этих подходов рассматриваются немного более подробно. В них не даны исчерпывающие формальные сведения, а указаны ссылки на литературу для дальнейшего изучения.

### **Методы на основе правил для формирования рассуждений в условиях неопределенности**

Системы на основе правил стали итогом ранних работ по практическим и интуитивным системам, применяемым для логического вывода. Логические системы в целом и логические системы на основе правил в частности обладают тремя описанными ниже желаемыми свойствами.

- **Локальность.** В логических системах, если имеется правило в форме  $A \Rightarrow B$ , можно вывести  $B$  при наличии свидетельства  $A$ , не учитывая существование каких-либо иных правил. С другой стороны, в вероятностных системах необходимо учитывать все свидетельства, представленные в марковском покрытии.

- **Отделение** (detachment). Как только будет найдено логическое доказательство для высказывания  $B$ , это высказывание может использоваться без учета того, как оно было получено. Это означает, что высказывание может быть **отделено** от его обоснования. С другой стороны, при манипулировании с вероятностями важно учитывать источник свидетельства, который служит обоснованием для рассматриваемой оценки степени уверенности, поскольку это требуется для дальнейшего формирования рассуждений.
- **Истинностная функциональность** (truth-functionality). В логике истинность сложных высказываний может быть вычислена на основании значений истинности их компонентов. Вероятностные комбинации не обладают таким свойством, за исключением тех случаев, когда используются сильные глобальные предположения о независимости.

В связи с этим было сделано несколько попыток разработать схемы неопределенных рассуждений, в которых сохранялись бы эти преимущества. При этом идея состояла в том, чтобы закрепить степени уверенности за высказываниями и правилами, а также разработать чисто локальные схемы для комбинирования и распространения таких степеней уверенности. Соответствующие схемы также обладают свойством истинностной функциональности; например, степень уверенности в истинности высказывания  $A \vee B$  является функцией от степени уверенности в истинности высказывания  $A$  и степени уверенности в истинности высказывания  $B$ .

Но перспективы успешного развития систем на основе правил находятся под сомнением в связи с тем, что свойства локальности, отделения и истинностной функциональности просто не распространяются на неопределенные рассуждения. Вначале рассмотрим истинностную функциональность. Допустим, что  $H_1$  является событием, связанным с тем, что подлинная монета падает орлом вверх, а также предположим, что  $T_1$  — событие, в котором эта монета при том же подбрасывании падает решкой вверх, а  $H_2$  — событие, что та же монета при втором подбрасывании падает орлом вверх. Безусловно, все три события имеют одну и ту же вероятность 0,5, поэтому система с истинностной функциональностью должна присваивать одну и ту же оценку степени уверенности дизъюнкции любых двух из этих событий. Но как показано в табл. 14.3, можно доказать, что вероятность дизъюнкции зависит от самих событий, а не только от их вероятностей.

**Таблица 14.3. Значения вероятностей отдельных событий и их дизъюнкций**

$P(A)$	$P(B)$	$P(A \vee B)$
	$P(H_1) = 0.5$	$P(H_1 \vee H_1) = 0.50$
$P(H_1) = 0.5$	$P(T_1) = 0.5$	$P(H_1 \vee T_1) = 1.00$
	$P(H_2) = 0.5$	$P(H_1 \vee H_2) = 0.75$

Ситуация становится еще хуже при соединении свидетельств в цепочку логических выводов. В системах с истинностной функциональностью предусмотрены **правила** в форме  $A \mapsto B$ , которые позволяют вычислять степень уверенности в истинности  $B$  как функцию от степени уверенности в истинности правила и степени уверенности в истинности  $A$ . Могут быть разработаны системы и прямого и обратного логического вывода. Предполагается, что степень уверенности в истинности правила

ла является постоянной, и это значение обычно определяется инженером по знаниям, например, как  $A \mapsto_{0.9} B$ .

Рассмотрим ситуацию с влажной травой, показанную на рис. 14.9, *a*. Если бы нам потребовалось иметь возможность проводить и причинные, и диагностические рассуждения, то нужны были бы два следующих правила:

$$\text{Rain} \mapsto \text{WetGrass} \text{ и } \text{WetGrass} \mapsto \text{Rain}$$

Эти два правила образуют петлю обратной связи: свидетельство в пользу события *Rain* повышает степень уверенности в истинности события *WetGrass*, что, в свою очередь, еще больше повышает степень уверенности в истинности события *Rain*. Очевидно, что в системах формирования неопределенных рассуждений приходится отслеживать пути, по которым распространяется свидетельство.

Сложными становятся также межпричинные рассуждения (или исключение из объяснений некоторых причин). Рассмотрим, что произойдет, если имеются следующие два правила:

$$\text{Sprinkler} \mapsto \text{WetGrass} \text{ и } \text{WetGrass} \mapsto \text{Rain}$$

Предположим, мы видим, что опрыскиватель включен. При формировании прямого логического вывода по нашим правилам это свидетельство увеличивает степень уверенности в том, что трава должна быть мокрой, а это, в свою очередь, увеличивает степень уверенности в том, что идет дождь. Но это же нелепо — тот факт, что опрыскиватель включен, полностью объясняет наличие мокрой травы и должен уменьшить степень уверенности в том, что идет дождь! А система с истинностной функциональностью действует таким образом, как будто в ней реализуется также уверенность в том, что применение опрыскивателя вызывает дождь, *Sprinkler*  $\mapsto$  *Rain*.

С учетом описанных сложностей, можно ли рассчитывать на то, что системы с истинностной функциональностью когда-либо найдут практическое применение? Ответ состоит в том, что для этого нужно ограничивать решаемые задачи и тщательно конструировать базу правил, чтобы в ней не возникали нежелательные взаимодействия. Наиболее известным примером системы с истинностной функциональностью для неопределенных рассуждений является модель **факторов определенности** (certainty factor), которая была разработана для программы медицинской диагностики *Mycin* и широко использовалась в экспертных системах в конце 1970-х и в 1980-х гг. Почти во всех областях использования факторов определенности предусматривались наборы правил, которые были либо чисто диагностическими (как в программе *Mycin*), либо чисто причинными. Более того, данные свидетельства вводились только в “корнях” мультидерева правил, а большинство деревьев правил были односвязными. Хекерман [639] показал, что при таких обстоятельствах незначительно модифицированный вариант вероятностного вывода на основе фактора определенности был точно эквивалентен байесовскому вероятностному выводу на полидеревьях. В других обстоятельствах применение факторов определенности из-за переоценки свидетельства приводило к появлению настолько неправильных степеней уверенности, что дальнейшая эксплуатация системы становилась невозможной. По мере увеличения размеров деревьев правил нежелательные взаимодействия правил становятся все более частыми, поэтому практики пришли к выводу, что приходится “поправлять” факторы определенности во многих других правилах после добавле-

ния новых правил. Нет смысла даже останавливаться на том, по каким причинам этот подход больше не рекомендуется.

### Представление незнания: теория Демпстера–Шефера

Теория  $\bowtie$  Демпстера–Шефера разработана для того, чтобы можно было учесть различие между **неопределенностью** (uncertainty) и **незнанием** (ignorance). Вместо вычисления вероятности высказывания в ней вычисляется вероятность того, что данное свидетельство поддерживает высказывание. Этот показатель измерения степени уверенности называется  $\bowtie$  доверительной функцией (belief function), которая обозначается как  $Bel(X)$ .

Вернемся к задаче с подбрасыванием монеты, чтобы рассмотреть пример применения доверительных функций. Предположим, что к вам подходит подозрительный тип и предлагает поспорить на 10 долларов, что его монета упадет орлом вверх после следующего подбрасывания. С учетом того, что эта монета может оказаться либо подлинной, либо фальшивой, какую степень уверенности вы должны назначить тому событию, что она упадет орлом вверх? В теории Демпстера–Шефера утверждается, что у вас пока нет свидетельств в пользу того или иного предположения, поэтому вам следует исходить из предположения, что степень уверенности  $Bel(Heads) = 0$  и что  $Bel(\neg Heads) = 0$ . Таким образом, в системах формирования рассуждений на основе теории Демпстера–Шефера принят скептический взгляд на то, что могут подсказывать некоторые интуитивные соображения. А теперь предположим, что вы пригласили эксперта, который освидетельствовал монету и с определенностью 90% заявил, что монета является подлинной (т.е. он на 90% уверен, что  $P(Heads) = 0.5$ ). В таком случае теория Демпстера–Шефера дает оценку  $Bel(Heads) = 0.9 \times 0.5 = 0.45$  и, аналогичным образом,  $Bel(\neg Heads) = 0.45$ . Таким образом, все еще остается “зазор” в 10%, который не охватывается этим свидетельством. В “правиле Демпстера” [382] показано, как комбинировать свидетельства для получения новых значений  $Bel$ , а в работе Шефера эти вычисления развиты до уровня полной вычислительной модели.

Однако, как и при формировании рассуждений по умолчанию, возникает проблема при соединении степеней уверенности с действиями. При использовании вероятностей теория принятия решений утверждает, что если  $P(Heads) = P(\neg Heads) = 0.5$ , то (при условии, что выигрыш в 10 долларов и проигрыш в 10 долларов рассматриваются как противоположные исходы с равной величиной) механизм формирования рассуждений не отдаст предпочтения одному из действий, в котором предложение сделать ставку отклоняется или принимается. С другой стороны, механизм формирования рассуждений Демпстера–Шефера имеет оценку  $Bel(\neg Heads) = 0$  и поэтому не видит причин принять ставку, но, кроме того, он имеет и оценку  $Bel(Heads) = 0$  и поэтому не видит причин, по которым он мог бы отклонить это предложение. Таким образом, на первый взгляд кажется, что механизм формирования рассуждений Демпстера–Шефера приходит к такому же заключению о том, как действовать в данном случае. К сожалению, теория Демпстера–Шефера не позволяет также прийти к определенному решению во многих других случаях, притом что в вероятностном выводе вырабатывается какой-то конкретный вариант. В действительности понятие полезности в модели Демпстера–Шефера еще не достаточно полно исследовано.

Еще в одной интерпретации теории Демпстера–Шефера предусмотрено определение интервала вероятностей, например, интервал для  $Heads$  равен  $[0, 1]$  до про-

верки монеты экспертом и становится равным  $[0.45, 0.55]$  после такой проверки. Оценка ширины этого интервала может способствовать выработке решения о получении дополнительных свидетельств; в частности, такая оценка говорит о том, что заявление эксперта поможет вам, если вы не знаете, является ли монета подлинной, но уже не поможет, если вы узнали, что монета — подлинная. Однако не существует четких рекомендаций в отношении того, как следует принимать указанные решения, поскольку еще нет четкого понимания того, что означает ширина интервала. При использовании байесовского подхода рассуждения такого рода могут быть сформированы более легко путем исследования вопроса, как изменилась бы степень уверенности некоего лица при получении им дополнительных свидетельств. Например, знание о том, что монета — подлинная, оказалось бы значительное влияние на степень уверенности в том, что она упадет орлом вверх, а обнаружение смещения ее центра тяжести оказало бы влияние на степень уверенности в том, что монета — подлинная. Полная байесовская модель должна была бы включать вероятностные оценки для факторов, подобных этому, что позволило бы нам выражать наше “незнание” в терминах того, как изменились бы значения степеней уверенности в процессе сбора информации в будущем.

### Представление неосведомленности: нечеткие множества и нечеткая логика

☞ **Теория нечетких множеств** — это один из подходов к определению того, насколько хорошо некоторый объект подходит под расплывчатое описание. Например, рассмотрим высказывание: “Нат имеет стройную фигуру”. Является ли это высказывание истинным, если Нат имеет рост примерно 180 см? Большинство людей затрудняются при выборе ответа “да” или “нет”, предпочитая сказать “скорее всего”. Обратите внимание на то, что этот вопрос не относится к сфере неопределенности знаний о внешнем мире — нам точно известен рост Ната. Проблема состоит в том, что лингвистическое выражение “стройный” не ссылается на четкую классификацию людей по признаку того, насколько стройной является его фигура. По этой причине ☞ *теория нечетких множеств вообще не является методом формирования неопределенных рассуждений*. Вместо этого в теории нечетких множеств оценка стройности фигуры, *Tall*, рассматривается как нечеткий предикат и принимается предположение, что истинностное значение высказывания *Tall(Nate)* — это число от 0 до 1, а не просто значение *true* или *false*. Само название “нечеткое множество” исходит из интерпретации предиката как неявно определяющего множество его элементов — множество, не имеющее четких границ.

☞ **Нечеткая логика** — это метод формирования рассуждений с помощью логических выражений, описывающих принадлежность элементов к нечетким множествам. Например, сложное высказывание *Tall(Nate)  $\wedge$  Heavy(Nate)* с оценкой того, насколько стройным является Нат и насколько велик его вес, имеет нечеткое истинностное значение, которое является функцией истинностных значений его компонентов. Ниже приведены стандартные правила оценки нечеткой истинности *T* сложного высказывания.

$$\begin{aligned} T(A \wedge B) &= \min(T(A), T(B)) \\ T(A \vee B) &= \max(T(A), T(B)) \\ T(\neg A) &= 1 - T(A) \end{aligned}$$

Поэтому нечеткая логика представляет собой систему с истинностной функциональностью, а этот факт становится причиной серьезных затруднений. Например, предположим, что  $T(\text{Tall}(\text{Nate})) = 0.6$  и  $T(\text{Heavy}(\text{Nate})) = 0.4$ . В таком случае получаем значение  $T(\text{Tall}(\text{Nate}) \wedge T(\text{Heavy}(\text{Nate}))) = 0.4$ , которое кажется обоснованным, но получаем также результат  $T(\text{Tall}(\text{Nate}) \wedge \neg \text{Tall}(\text{Nate})) = 0.4$ , который таковым не кажется. Безусловно, эта проблема возникает из-за того, что истинностно-функциональный подход не позволяет учитывать корреляции или антикорреляции между компонентами высказываний.

↗ **Нечеткое управление** — это методология создания систем управления, в которых отображение между реальными входными данными и выходными параметрами представлено с помощью нечетких правил. Нечеткое управление оказалось очень успешным в таких коммерческих продуктах, как автоматические коробки передач, видеокамеры и электрические бритвы. Критики этого подхода утверждают, что такие приложения оказались успешными потому, что в них используются небольшие базы правил, логические выводы не формируют цепочки, а для повышения производительности системы может осуществляться настройка параметров (см., например, [434]). И действительно, тот факт, что правила функционирования этих систем реализованы с помощью нечетких операторов, может не быть ключевым фактором их успеха; секрет состоит в том, чтобы применялся лаконичный и интуитивно понятный способ задания гладко интерполируемой функции с реальными значениями.

Предпринимались также попытки предложить одну из трактовок нечеткой логики в терминах теории вероятностей. Одна из таких идей состоит в том, чтобы такие утверждения, как “Нат — стройный”, рассматривались в качестве дискретных наблюдений, сделанных применительно к непрерывной скрытой переменной — фактическому росту, *Height*, Ната. В такой вероятностной модели определяется вероятность  $P(\text{Наблюдатель сообщает, что Нат — стройный} | \text{Height})$ , возможно, с использованием **пробит-распределения** (см. с. 674). В таком случае появляется возможность рассчитать апостериорное распределение вероятностей значений роста Ната обычным образом, например, если эта модель входит в состав гибридной байесовской сети. Безусловно, такой подход не относится к категории истинностно-функциональных. Например, следующее условное распределение:

$$P(\text{Наблюдатель сообщает, что Нат имеет стройную фигуру и большой вес} | \text{Height}, \text{Weight})$$

допускает взаимодействия роста и веса в обоснованиях наблюдения. Таким образом, некто, имеющий рост примерно 245 см и вес 85 кг, с очень малой вероятностью будет назван “имеющим стройную фигуру и большой вес”, даже несмотря на то, что человек, имеющий рост “245 см” рассматривается как “стройный”, а вес “85 кг” принято считать “большим”.

Нечетким предикатам может быть также дана вероятностная интерпретация в терминах ↗ **случайных множеств**, т.е. случайных переменных, возможными значениями которых являются множества объектов. Например, *Tall* — это случайное множество, возможными значениями которого являются множества людей. Вероятность  $P(\text{Tall} = S_1)$ , где  $S_1$  — некоторое определенное множество людей, представляет собой вероятность того, что именно данное множество будет обозначено некоторым наблюдателем как “характеризующееся стройными фигурами”. Таким образом, вероятность того, что “Нат имеет стройную фигуру”, представляет собой сумму вероятностей для всех множеств, элементом которых является Нат.

Создается впечатление, что и подход на основе гибридных байесовских сетей, и подход с использованием случайных множеств охватывает многие аспекты нечеткости, не требуя учета степеней истинности. Тем не менее остается много нерешенных вопросов, касающихся правильного представления лингвистических наблюдений и непрерывных количеств, а этими вопросами пренебрегают большинство исследователей, не относящихся к сообществу специалистов по нечетким представлениям.

## 14.8. РЕЗЮМЕ

В этой главе описаны **байесовские сети** — тщательно разработанное представление для неопределенных знаний. Байесовские сети играют роль, примерно аналогичную той, которую выполняет пропозициональная логика применительно к определенным знаниям.

- Байесовская сеть — это ориентированный ациклический граф, вершины которого соответствуют случайным переменным; с каждой вершиной связано распределение условных вероятностей для этой вершины, если даны ее родительские вершины.
- Байесовские сети лежат в основе удобного способа представления отношений **условной независимости** в рассматриваемой проблемной области.
- Любая байесовская сеть задает полное совместное распределение; каждый элемент совместного распределения определяется как произведение соответствующих элементов в локальных условных распределениях. Байесовская сеть часто позволяет экспоненциально уменьшить размеры вероятностного представления по сравнению с полным совместным распределением.
- Многие условные распределения могут быть представлены компактно с помощью канонических семейств распределений. Целый ряд канонических распределений используется в **гибридных байесовских сетях**, которые включают и дискретные, и непрерывные переменные.
- Под вероятностным выводом в байесовских сетях подразумевается вычисление распределения вероятностей множества переменных запроса, если дано множество переменных свидетельства. Алгоритмы точного вероятностного вывода, такие как алгоритм **устранения переменной**, позволяют вычислять суммы произведений условных вероятностей настолько эффективно, насколько это возможно.
- В **полидеревых** (односвязных сетях) точный вероятностный вывод требует времени, линейно зависящего от размера сети. А в общем случае проблема такого вывода неразрешима.
- Методы стохастической аппроксимации, такие как **оценка веса с учетом правдоподобия** и метод **Монте-Карло на основе цепи Маркова**, позволяют получить приемлемые оценки истинных апостериорных вероятностей в сети и способны справиться с гораздо более крупными сетями по сравнению с точными алгоритмами.

- Теория вероятностей может применяться в сочетании с идеями представления знаний, заимствованными из логики первого порядка, для создания очень мощных систем формирования рассуждений в условиях неопределенности. **Реляционные модели вероятностей** (Relational Probability Model — RPM) включают ограничения на средства представления, которые гарантируют получение полностью определенного распределения вероятностей, которое может быть выражено в виде эквивалентной байесовской сети.
- Был предложен целый ряд альтернативных систем для формирования рассуждений в условиях неопределенности. Вообще говоря, **истинностно-функциональные** системы не очень хорошо подходят для таких рассуждений.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Широкое использование сетей для представления вероятностной информации началось с первых десятилетий XX столетия, когда были опубликованы работы Сьюэлла Райта по вероятностному анализу генетического наследования и показателей развития животных [1622], [1624]. Одна из его сетей показана на обложке данной книги. И.Дж. Гуд [575] в сотрудничестве с Аланом Тьюрингом разработал вероятностные представления и методы байесовского вероятностного вывода, которые могут рассматриваться как предшествующие современным байесовским сетям, хотя указанная статья не часто цитируется в данном контексте<sup>10</sup>. Та же статья является оригинальным литературным источником с описанием модели зашумленного OR.

Форма представления для задач принятия решений с помощью **диаграммы влияния**, которая была встроена в представление DAG для случайных переменных, использовалась в анализе принятия решений с конца 1970-х годов (глава 16), но для вычислений применялись только методы перебора. Джуди Перл разработал метод передачи сообщений для осуществления вероятностного вывода в древовидных сетях [1186] и ввел понятие полидревовидных сетей [796], а также объяснил важность составления причинных, а не диагностических вероятностных моделей, в противовес системам, основанным на использовании факторов определенности, которые были в моде в то время. Первой экспертной системой, в которой использовались байесовские сети, стала Convince [795], [797]. Многие новейшие медицинские системы включают систему Munin для диагностирования нейромускульных нарушений [27] и систему Pathfinder для выявления патологий [640]. Одними из наиболее широко используемых систем на основе байесовских сетей оказались модули диагностики и восстановления (например, модуль Printer Wizard) в операционной системе Microsoft Windows [178] и Office Assistant в пакете Microsoft Office [685].

Перл [1189] разработал алгоритм кластеризации для точного вероятностного вывода в байесовских сетях общего вида, используя метод преобразования в ориентированное полидрево кластеров, в котором для достижения согласованности по переменным, разделяемым между кластерами, использовалась передача сообщений.

<sup>10</sup> И. Дж. Гуд был главным статистиком в группе Тьюринга, занимавшейся раскрытием шифров во время Второй мировой войны. В книге 2001: A Space Odyssey [264] выражена благодарность Гуду и Минскому за их вклад в тот научный прорыв, который привел к разработке компьютера HAL 9000.

Аналогичный подход, разработанный статистиками Дэвидом Шпигельхальтером и Штеффеном Лауритценом [895], [1449], основан на преобразовании в неориентированную сеть (Маркова). Этот подход реализован в системе Hugin, которая представляет собой эффективное и широко применяемое инструментальное средство для формирования рассуждений в условиях неопределенности [27]. Росс Шахтер, работающий в сообществе исследователей диаграмм влияния, разработал точный метод, который основан на сокращении сети, управляемом целями, с использованием преобразований, сохраняющих апостериорную вероятность [1383].

Метод устранения переменных, описанный в данной главе, ближе всего по смыслу к методу Шахтера, на основе которого разработан алгоритм символического вероятностного вывода (Symbolic Probabilistic Inference — SPI) [1385]. В алгоритме SPI предпринимается попытка оптимизировать вычисление деревьев выражений, подобных приведенному на рис. 14.8. Описанный в данной книге алгоритм больше всего напоминает алгоритм, разработанный Чжантом и Пулом [1643], [1644]. Критерии отсечения нерелевантных переменных были разработаны Гейгером и др. [530], а также Лауритценом и др. [894]; приведенный в данной книге критерий представляет собой простой частный случай этих критериев. Рина Дехтер [369] показала, что идея устранения переменной по сути идентична ~~непоследовательному динамическому программированию~~ [117] — алгоритмическому подходу, который может использоваться для решения целого ряда задач вероятностного вывода в байесовских сетях, например поиска **наиболее вероятного объяснения** для множества наблюдений. В этом подходе алгоритмы байесовских сетей применяются в сочетании с соответствующими методами решения задач CSP и дается прямая оценка меры сложности точного вывода в терминах **ширины гипердерева** сети.

Вопрос о включении непрерывных случайных переменных в байесовские сети рассматривался Перлом [1191], а также Шахтером и Кенли [1386]; в этих статьях обсуждаются сети, содержащие только непрерывные переменные с линейными гауссовыми распределениями. Проблема включения дискретных переменных исследовалась Лауритценом и Вермутом [896], а полученные результаты реализованы в системе cHUGIN [1154]. Пробит-распределение впервые было исследовано Финнеем [469], который назвал его сигмоидальным распределением. Это распределение широко использовалось для моделирования феномена дискретного выбора и может быть дополнено, если требуется его применение в тех случаях, когда количество выборов превышает два [318]. В [133] приведено обоснование целесообразности использования логит-распределения в данной научной области.

Купер [291] показал, что общая проблема вероятностного вывода в байесовских сетях без ограничений является NP-трудной, и Пол Дагум и Майк Луби [319] показали, что NP-трудной является соответствующая задача аппроксимации. Одной из серьезных проблем при использовании методов кластеризации и устранения переменной является также пространственная сложность. Применение метода **определения условий выбора множества разрыва цикла**, который был разработан для задач CSP в главе 5, позволяет исключить необходимость построения экспоненциально больших таблиц. В байесовской сети множеством разрыва цикла является множество вершин, позволяющих после их конкретизации свести оставшиеся вершины к полидереву, которое может быть решено с линейными затратами времени и пространства. Поиск ответа на запрос осуществляется путем суммирования по всем конкретизациям множества разрыва цикла, поэтому общие требования к пространству все

еще остается линейными [1191]. В [323] описан рекурсивный алгоритм обусловливания, который допускает широкий выбор компромиссных методов сокращения затрат пространства/времени.

В настоящее время разработка быстрых алгоритмов аппроксимации для вероятностного вывода в байесовских сетях представляет собой очень активную научную область, которая испытывает положительное влияние со стороны статистики, компьютерных наук и физики. Способ формирования выборок с исключением представляет собой общий метод, давно известный статистикам; он был впервые применен к байесовским сетям Максом Хенрионом [648], который назвал этот метод **логическим формированием выборок**. Метод взвешивания с учетом правдоподобия, который был разработан Фунгом и Чангом [512], а также Шахтером и Пеотом [1387], представляет собой пример широко известного статистического метода **формирования выборок с учетом важности**. Результаты крупномасштабного применения метода взвешивания с учетом правдоподобия в области медицинской диагностики опубликованы в [1407]. Чэнг и Друдзель [247] описали адаптивную версию метода взвешивания с учетом правдоподобия, которая действует очень успешно, даже если свидетельства имеют крайне низкое априорное правдоподобие.

Развитие алгоритмов Монте-Карло на основе цепи Маркова (Markov chain Monte Carlo — MCMC) началось с создания алгоритма Метрополиса, впервые опубликованного в статье [1036], которая стала также первой публикацией сведений об алгоритме эмуляции отжига (см. главу 4). Формирователь выборок Гиббса был предложен в [535] для вероятностного вывода в неориентированных сетях Маркова. Применение алгоритма MCMC к байесовским сетям было предложено Перлом [1190]. Статьи, собранные в [554], охватывают широкий диапазон направлений использования алгоритма MCMC, многие из которых были разработаны при создании широко известного пакета Bugs [555].

В этой главе не рассматривались два очень важных семейства методов аппроксимации. Первым из них является семейство методов **вариационной аппроксимации**, которые могут использоваться для упрощения сложных вычислений любых типов. Основная идея состоит в том, что должна быть предложена сокращенная версия первоначальной задачи, с которой легче работать, но которая напоминает первоначальную задачу настолько близко, насколько это возможно. Сокращенная задача описывается с помощью некоторых **вариационных параметров**  $\lambda$ , которые корректируются с целью минимизации функции расстояния  $D$  между оригинальной и сокращенной задачами, часто путем решения системы уравнений  $\partial D / \partial \lambda = 0$ . Во многих случаях могут быть получены строгие верхние и нижние границы. Вариационные методы уже давно использовались в статистике [1333]. В статистической физике метод  **поля осредненных величин** (mean field) представляет собой особую вариационную аппроксимацию, в которой предполагается, что отдельные переменные, входящие в состав модели, являются полностью независимыми. Эта идея была применена для поиска решений в крупных неориентированных сетях Маркова [1174], [1209]. В [1353] представлены результаты разработки математических основ применения вариационных методов к байесовским сетям и получения точных аппроксимаций нижней границы для сигмоидальных сетей с использованием методов поля осредненных величин. Эта методология была дополнена в [720] для получения нижней и верхней границ. Обзор вариационных подходов приведен в [748].

Второе важное семейство алгоритмов аппроксимации основано на алгоритме Перла передачи сообщений в полидереве [1186]. Как указал Перл [1191], этот алгоритм может применяться к сетям общего типа. Иногда результаты оказываются неправильными, или же не удается добиться нормального завершения работы алгоритма, но во многих случаях полученные значения близки к истинным значениям. Так называемый подход с **распространением оценок степени уверенности** (или с **циклическим распространением**) привлекал мало внимания до тех пор, пока Макэлис и др. [1027] не обнаружили, что передача сообщений в многосвязной байесовской сети полностью аналогична вычислениям, выполняемым в алгоритме **турбодекодирования** (turbo decoding) [115], который оказался крупным научным прорывом в области разработки эффективных кодов коррекции ошибок. Из этого следует вывод, что способ циклического распространения быстро и точно работает в очень крупных и тесно связанных сетях, используемых для декодирования, и поэтому может найти более широкое применение. В [1105] представлены результаты эмпирического исследования областей использования этого алгоритма. В [1630] подробно описаны связи между методом циклического распространения и идеями статистической физики.

Связь между вероятностными методами и языками первого порядка была впервые исследована Карнапом [225]. В [513] и [1373] определен язык, в котором вероятности могут быть объединены с высказываниями первого порядка и для которого моделями являются вероятностные показатели в возможных мирах. В рамках искусственного интеллекта эта идея была разработана Нильссоном для пропозициональной логики [1144] и Халперном для логики первого порядка [607]. Результаты первого обширного исследования проблем представления знаний в таких языках были опубликованы в [51], а в [1577] приведен обзор первых подходов к реализации этих способов представления на основе формирования эквивалентных пропозициональных байесовских сетей. В дальнейшем исследователи пришли к пониманию важности полных баз знаний, т.е. баз знаний, в которых, как и в байесовских сетях, определяется уникальное совместное распределение по всем возможным мирам. Методы осуществления этого были основаны на вероятностных версиях логического программирования [1226], [1352] или на семантических сетях [823]. Реляционные вероятностные модели такого типа, описанные в данной главе, были глубоко исследованы Пфеффером [1210]. В [1179] приведены результаты исследования проблем реляционной неопределенности и неопределенности идентичности в рамках моделей RPM, а также результаты использования вероятностного вывода с помощью алгоритма MCMC.

Как было описано в главе 13, после появления первых вероятностных систем в начале 1970-х годов интерес к ним упал и частично образовался вакуум, который стали заполнять альтернативные методы. Для использования в медицинской экспертной системе Mycin [1406] был разработан подход на основе факторов определенности, который был предназначен как для выработки проектных решений, так и для моделирования субъективных суждений, формируемых в условиях неопределенности. В сборнике статей *Rule-Based Expert Systems* [204] приведен полный обзор системы Mycin и других систем, разработанных на ее основе (см. также [1458]). Дэвид Хекерман [639] показал, что в некоторых случаях слегка модифицированная версия на основе вычислений с фактором определенности позволяет получить правильные вероятностные результаты, но в других случаях приводит к серьезной переоценке важности свидетельств. В экспертной системе Prospector [420] используется подход на

основе правил, в котором правила были обоснованы с помощью предположения о глобальной независимости (которое редко оправдывается).

Теория, получившая название теории Демпстера–Шефера, была впервые опубликована в статье Артура Демпстера [382], который предложил обобщение способа представления вероятностей в виде интервальных значений и обосновал правила комбинирования для их использования. После того как в дальнейшем была опубликована работа Гленна Шефера [1388], теория Демпстера–Шефера стала рассматриваться как научный подход, способный конкурировать с вероятностным подходом. В [1319] приведены результаты анализа связи между теорией Демпстера–Шефера и стандартной теорией вероятностей. Шеной [1401] предложил метод принятия решений с помощью доверительных функций Демпстера–Шефера.

Нечеткие множества были разработаны Лотфи Задэ [1637] в целях устранения широко признанных сложностей при предоставлении точных входных данных для интеллектуальных систем. В [1649] приведено исчерпывающее введение в теорию нечетких множеств; статьи по приложениям нечетких множеств собраны в [1648]. Как уже было указано в данной книге, нечеткую логику часто трактуют неправильно, усматривая в ней прямого конкурента для теории вероятностей, тогда как в этой теории фактически рассматриваются другие вопросы. Для вычислений с учетом неопределенности в нечетких системах была предложена **теория возможностей** (possibility theory) [1638], которая имеет много общего с теорией вероятностей. В [419] предложен исчерпывающий обзор по проблеме связей между теорией возможностей и теорией вероятностей.

Пробуждение в последнее время интереса к вероятностным методам в основном вызвано тем открытием, что байесовские сети являются удобным средством представления и использования информации об условной независимости. Но сторонникам байесовского подхода пришлось бороться за его дальнейшее распространение; некоторое представление о том, какие дебаты они вели со своими противниками, можно получить, ознакомившись с воинственной статьей Питера Чизмена *In Defense of Probability* [242] и с его последующей статьей *An Inquiry into Computer Understanding* [243] (с комментариями). Одним из принципиальных возражений противников байесовского подхода (и последователей логицистского подхода) было то, что числовые вычисления, применяемые в теории вероятностей, не очевидны для интуитивного восприятия и претендуют на наличие нереального уровня точности в наших неопределенных знаниях. Результаты разработки Веллманом **качественных вероятностных сетей** [1574] привели к созданию чисто качественной абстракции байесовских сетей, в которой используется понятие положительных и отрицательных влияний между переменными. Веллман показал, что во многих случаях такая информация является достаточной для принятия оптимальных решений и не требует точного указания вероятностных значений. В работе Эднана Дарвича и Мэтта Гинсберга [324] выявлены основные свойства методов обусловливания и комбинирования свидетельств, разработанных в рамках теории вероятностей, и показано, что эти методы могут также применяться при формировании логических рассуждений и рассуждений по умолчанию.

Система диагностики сердечных заболеваний, описанная в данной главе, разработана Лукасом [962]. К другим отраслевым приложениям байесовских сетей относится выполненные в компании Microsoft работы по выявлению целей пользователя компьютера на основании анализа его действий [685] и по фильтрации нежелательной электронной почты [1344], а также работа Electric Power Research Institute по

контролю над электрическими генераторами [1088] и работа NASA по выводу крайне чувствительной ко времени информации на дисплей в Центре управления полетами в Хьюстоне [684].

Некоторые важные ранние статьи по применению методов формирования рассуждений в условиях неопределенности в искусственном интеллекте собраны в антологиях *Readings in Uncertain Reasoning* [1389] и *Uncertainty in Artificial Intelligence* [768]. Наиболее важной отдельной публикацией, посвященной развитию байесовских сетей, безусловно, является книга *Probabilistic Reasoning in Intelligent Systems* [1191]. Более свежие материалы можно найти в нескольких превосходных книгах, включая [734], [746] и [893]. Новейшие результаты исследований в области вероятностных рассуждений публикуются и в профильных журналах по искусственному интеллекту, таких как *Artificial Intelligence* и *Journal of AI Research*, и в более специализированных журналах, таких как *International Journal of Approximate Reasoning*. Многие статьи по **графическим моделям**, к которым относятся байесовские сети, публикуются в статистических журналах. Превосходными источниками сведений о современных исследованиях являются труды конференций *Uncertainty in Artificial Intelligence* (UAI), *Neural Information Processing Systems* (NIPS) и *Artificial Intelligence and Statistics* (AISTATS).

## УПРАЖНЕНИЯ

**14.1.** Рассмотрите сеть для диагностики автомобиля, показанную на рис. 14.12.

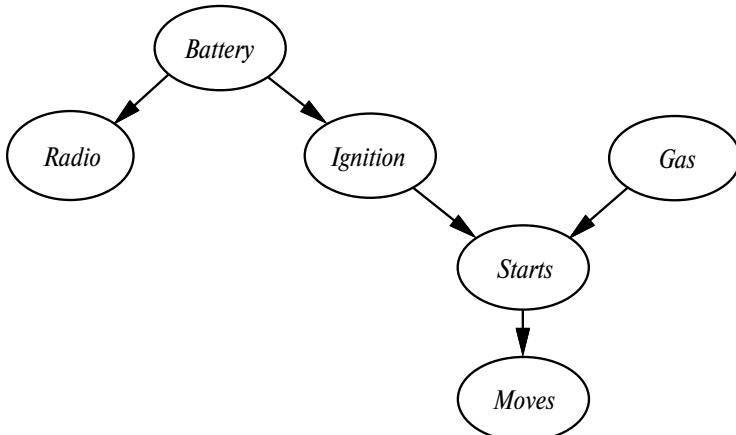


Рис. 14.12. Байесовская сеть, описывающая некоторые характеристики электрической системы и двигателя автомобиля. Каждая переменная является булевой, а значение true указывает на то, что соответствующая подсистема автомобиля находится в рабочем состоянии

- a) Дополните сеть булевыми переменными *IcyWeather* (Морозная погода) и *StarterMotor* (Стартер).
- б) Приведите приемлемые таблицы условных вероятностей для всех вершин.
- в) Сколько независимых значений содержится в совместном распределении вероятностей для восьми булевых вершин, если исходить из предположе-

ния, что неизвестны какие-либо отношения условной независимости, которые бы их связывали?

- г) Какое количество независимых вероятностных значений содержится в таблицах вашей сети?
  - д) Условное распределение для вершины *Starts* (Запускается) может быть описано как распределение **зашумленного AND**. Дайте общее определение этого семейства распределений и покажите его связь с распределениями зашумленного OR.
- 14.2.** Допустим, что вы живете рядом с атомной электростанцией, в которой предусмотрена тревожная сигнализация, срабатывающая, если показания датчика температуры превышают некоторое пороговое значение. Датчик измеряет температуру в реакторе. Рассмотрите булевые переменные  $A$  (звукит тревожный сигнал),  $F_A$  (тревожная сигнализация неисправна) и  $F_G$  (неисправен датчик), а также многозначные вершины  $G$  (показания датчика) и  $T$  (фактическая температура в реакторе).
- а) Нарисуйте байесовскую сеть для этой проблемной области с учетом того, что вероятность отказа датчика повышается, если температура в реакторе становится слишком высокой.
  - б) Является ли ваша сеть полидеревом?
  - в) Примите предположение, что есть только два значения фактической и измеряемой температур — нормальная и высокая; вероятность того, что датчик сообщает правильную температуру, равна  $x$ , когда он работает, а если не исправен, равна  $y$ . Приведите таблицу условных вероятностей, связанную с вершиной  $G$ .
  - г) Примите предположение, что тревожная сигнализация действует правильно, при условии, что она не вышла из строя; в последнем случае тревожный сигнал никогда не звучит. Приведите таблицу условных вероятностей, связанную с вершиной  $A$ .
  - д) Примите предположение, что тревожная сигнализация и датчик исправны и что тревожный сигнал звучит. Вычислите выражение для вероятности того, что температура в реакторе слишком высока, в терминах различных условных вероятностей в данной сети.
- 14.3.** Два астронома в различных частях Земли провели измерения,  $M_1$  и  $M_2$ , количества звезд,  $N$ , в некотором небольшом регионе неба с помощью своих телескопов. Обычно вероятность ошибки  $e$  на одну звезду в большую или меньшую сторону при подсчете этого количества не очень велика. Каждый телескоп может также оказаться (с гораздо меньшей вероятностью  $f$ ) сильно расфокусированным (события  $F_1$  и  $F_2$ ), и в этом случае ученый не досчитается трех или большего количества звезд (или, если  $N$  меньше 3, вообще не обнаружит ни одной звезды). Рассмотрите эти три сети, показанные на рис. 14.13.
- а) Какая из этих трех байесовских сетей может служить правильным (но не обязательно эффективным) представлением приведенной выше информации?
  - б) Какая из этих сетей является самой лучшей? Объясните, почему.

- в) Запишите распределение условных вероятностей  $P(M_1 | N)$  для случая, где  $N \in \{1, 2, 3\}$  и  $M_1 \in \{0, 1, 2, 3, 4\}$ . Каждая запись в распределении условных вероятностей должна быть представлена как функция от параметров  $e$  и/или  $f$ .
- г) Примите предположение, что  $M_1=1$  и  $M_2=3$ . Каково возможное количество звезд, если предполагается, что на значения  $N$  не налагаются априорные ограничения?
- д) Каково наиболее вероятное количество звезд, если даны эти наблюдения? Объясните, как рассчитать эту вероятность, или, если ее невозможно рассчитать, объясните, какая требуется дополнительная информация и как она влияет на результат.
- 14.4.** Рассмотрите сеть, показанную на рис. 14.13, б; примите предположение, что два телескопа дают идентичные результаты,  $N \in \{1, 2, 3\}$  и  $M_1, M_2 \in \{0, 1, 2, 3, 4\}$ , а также, что применяются такие символические таблицы СРТ, как описано в упр. 14.3. Используя алгоритм перебора, рассчитайте распределение вероятностей  $P(N | M_1=2, M_2=2)$ .

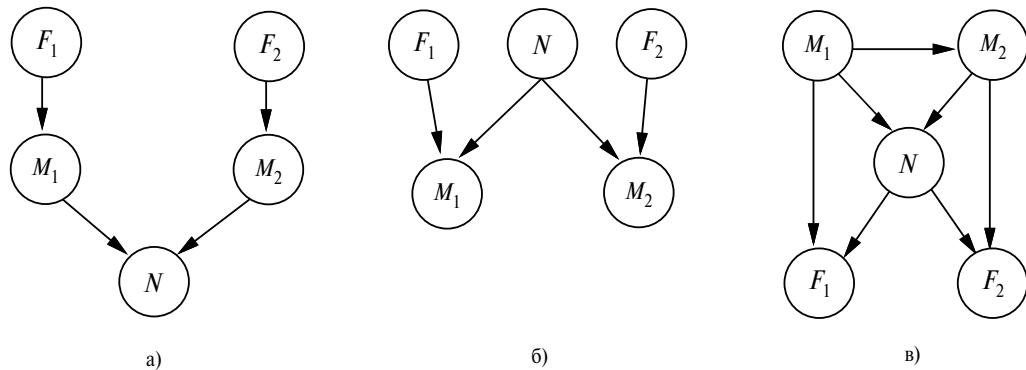


Рис. 14.13. Три возможные сети для задачи с телескопом

- 14.5.** Рассмотрите семейство линейных гауссовых сетей, которое показано на с. 672.
- а) Допустим, что в сети с двумя переменными переменная  $X_1$  является родительской по отношению к переменной  $X_2$ , переменная  $X_1$  имеет гауссово распределение априорных вероятностей, а  $P(X_2 | X_1)$  — линейное гауссово распределение. Покажите, что совместное распределение  $P(X_1, X_2)$  представляет собой многомерное гауссово распределение, и рассчитайте его матрицу ковариации.
- б) Докажите по индукции, что совместное распределение для линейной гауссовой сети общего вида по переменным  $X_1, \dots, X_n$  также является многомерным гауссовым распределением.
- 14.6.** Пробит-распределение, описанное на с. 674, позволяет представить распределение вероятностей для булевой дочерней переменной, если задана одна непрерывная родительская переменная.

- a) Как можно расширить это определение, чтобы оно охватывало несколько непрерывных родительских переменных?
- б) Как оно может быть расширено с учетом многозначной дочерней переменной? Рассмотрите и те случаи, в которых значения дочерней переменной упорядочены (как при выборе во время вождения автомобиля передачи в зависимости от скорости, крутизы подъема, требуемого ускорения и т.д.), и те случаи, в которых они не упорядочены (как при выборе для проезда на работу или автобуса, или электропоезда, или автомобиля). (*Подсказка.* Рассмотрите способы распределения возможных значений по двум множествам для имитации булевой переменной.)
- 14.7.** В этом упражнении речь идет об алгоритме устраниния переменной, приведенном в листинге 14.2.
- a) В разделе 14.4 алгоритм устраниния переменной применялся к следующему запросу:  
 $P(\text{Burglary} \mid \text{JohnCalls}=\text{true}, \text{MaryCalls}=\text{true})$
- Проведите указанные вычисления и проверьте правильность ответа.
- б) Подсчитайте количество выполненных арифметических операций и сравните его с количеством операций, выполняемых в алгоритме перебора.
- в) Предположим, что сеть имеет форму цепи — последовательности булевых переменных  $X_1, \dots, X_n$ , где  $\text{Parents}(X_i) = \{X_{i-1}\}$  для  $i=2, \dots, n$ . Какова сложность вычисления выражения  $P(X_1 \mid X_n=\text{true})$  с использованием алгоритма перебора? С использованием алгоритма устраниния переменной?
- г) Докажите, что сложность применения алгоритма устраниния переменной в сети, имеющей форму полидерева, линейно зависит от размера дерева при любом упорядочении переменных, согласованном со структурой сети.
- 14.8.** Проведите исследование сложности точного вывода в байесовских сетях общего вида.
- a) Докажите, что любую задачу 3-SAT можно свести к задаче точного вывода в байесовской сети, сформированной для представления данной конкретной задачи, и поэтому такой точный вывод является NP-трудным. (*Подсказка.* Рассмотрите сеть с одной переменной для каждого пропозиционального символа, с одной для каждого выражения и с одной для конъюнкции выражений.)
- б) Проблема подсчета количества выполняющих присваиваний для задачи 3-SAT является #P-полной (шарп-, или диэз-P-полной). Покажите, что задача точного вывода является, по меньшей мере, такой же трудной, как эта.
- 14.9.** Рассмотрите задачу формирования случайной выборки из заданного распределения по одной переменной. Вы можете принять предположение, что в вашем распоряжении имеется генератор случайных чисел, который возвращает случайное число с равномерным распределением в интервале от 0 до 1.
- a) Допустим, что  $X$  — дискретная переменная с  $P(X=x_i)=p_i$  для  $i \in \{1, \dots, k\}$ .  **Кумулятивное распределение** (cumulative distribution)  $X$  задает вероятность того, что  $X \in \{x_1, \dots, x_j\}$  для каждого возможного  $j$ . Объясните, как рассчитать это кумулятивное распределение за время

$O(k)$  и как получить из него одну выборку  $X$ . Может ли последняя операция быть выполнена за время меньше  $O(k)$ ?

- Теперь предположим, что необходимо сформировать  $N$  выборок  $X$ , где  $N \gg k$ . Объясните, как выполнить эту операцию с ожидаемым временем прогона в расчете на выборку, которое является постоянным (т.е. независимым от  $k$ ).
- После этого рассмотрим непрерывную переменную с параметризованным (например, с гауссовым) распределением. Как можно формировать выборки с помощью такого распределения?
- Предположим, что необходимо сформулировать запрос, касающийся непрерывной переменной, и что для вероятностного вывода используется такой алгоритм, как *LikelihoodWeighting*. Как вы модифицировали бы процесс поиска ответов на такие запросы?

#### 14.10. Марковское покрытие

- переменной определено на с. 669.
- Докажите, что переменная является независимой от всех других переменных в сети, если дано ее марковское покрытие.
  - Выполните уравнение 14.11.

#### 14.11. Рассмотрите запрос $\mathbf{P}(\text{Rain} | \text{Sprinkler}=\text{true}, \text{WetGrass}=\text{true})$

- к байесовской сети, приведенной на рис. 14.9, а, и покажите, как можно получить на него ответ с помощью алгоритма МCMC.
- Какое количество состояний имеет эта цепь Маркова?
  - Рассчитайте матрицу переходов  $Q$ , содержащую значение  $q(y \rightarrow y')$  для всех  $y, y'$ .
  - Что представляет собой  $Q^2$ , квадрат матрицы переходов?
  - А что можно сказать о выражении  $Q^n$ , где  $n \rightarrow \infty$ ?
  - Объясните, как следует выполнять вероятностный вывод в байесовских сетях, при условии, что доступно выражение  $Q^n$ . Является ли такой способ вероятностного вывода практически применимым?

#### 14.12. Три футбольные команды, $A$ , $B$ и $C$ , играют друг с другом по одному разу. Каждый матч проводится между двумя командами и может закончиться для команды выигрышем, ничьей или проигрышем. Каждой команде присвоена постоянная, неизвестная оценка ее класса (целое число в диапазоне от 0 до 3), и результат матча оценивается вероятностной зависимостью от разницы в классе между двумя участвующими командами.

- Сформируйте реляционную вероятностную модель для описания этой проблемной области и предложите реальные числовые значения для всех необходимых распределений вероятностей.
- Сформируйте эквивалентную байесовскую сеть.
- Примите предположение, что в первых двух матчах команда  $A$  побеждает  $B$  и играет вничью с  $C$ . Используя выбранный вами алгоритм точного вывода, вычислите распределение апостериорных вероятностей для результатов третьего матча.

- г) Предположим, что в этой футбольной лиге  $n$  команд и известны результаты всех матчей, кроме последнего. Как сложность предсказания результатов последней игры зависит от  $n$ ?
- д) Рассмотрите возможность применения алгоритма МСМС для решения этой задачи. Насколько быстро этот алгоритм сходится на практике и насколько хорошо он масштабируется?

# 15 ВЕРОЯТНОСТНЫЕ РАССУЖДЕНИЯ ВО ВРЕМЕНИ

*В данной главе предпринимается попытка столкнуть настоящее, понять прошлое, а также, возможно, предсказать будущее, даже если пока лишь немногое полностью ясно.*

Агенты, действующие в неопределенных вариантах среды, должны быть способны следить за текущим состоянием среды; это требование аналогично тому, которое предъявляется к логическим агентам. Указанная задача в значительной степени усложняется из-за наличия частичных и зашумленных результатов восприятия, а также из-за неопределенности в отношении того, как среда изменяется во времени. В лучшем случае агент будет способен получить только вероятностную оценку текущей ситуации. В этой главе описаны способы представления и алгоритмы вероятностного вывода, благодаря которым такая оценка становится возможной, на основе идей, представленных в главе 14.

Базовый подход описан в разделе 15.1: изменяющийся мир моделируется с использованием случайных переменных, обозначающих каждый аспект состояния мира в каждый момент времени. Отношения между этими переменными описывают, как развивается данное состояние. В разделе 15.2 определены основные задачи вероятностного вывода и описана общая структура алгоритмов вероятностного вывода для временных моделей. Затем рассматриваются три конкретных типа моделей: **скрытые марковские модели, фильтры Калмана и динамические байесовские сети** (которые включают скрытые марковские модели и фильтры Калмана в качестве частных случаев). Наконец, в разделе 15.6 показано, благодаря чему временные вероятностные модели образуют ядро современных систем распознавания речи. Центральную роль в создании всех этих моделей играет обучение, но подробное исследование алгоритмов обучения будет отложено до части VI.

## 15.1. ВРЕМЯ И НЕОПРЕДЕЛЕННОСТЬ

В предыдущих главах методы вероятностных рассуждений рассматривались в контексте **статических** миров, в которых каждая случайная переменная имеет одно фиксированное значение. Например, в ходе ремонта автомобиля предполагается, что неисправный узел остается неисправным на протяжении всего процесса диагностики; наша задача состоит в вероятностном выводе информации о состоянии автомобиля из наблюдаемых свидетельств, которые также остаются фиксированными.

Теперь обратимся к несколько другой задаче — лечение больного диабетом. Как и в случае ремонта автомобиля, в нашем распоряжении имеются факты, такие как последние дозы приема инсулина, рацион питания, результаты измерения количества сахара в крови и другие физические симптомы. Задача состоит в том, чтобы оценить текущее состояние пациента, включая фактический уровень сахара в крови и уровень инсулина. При наличии такой информации врач (или больной) принимает решение о рационе питания больного и дозе инсулина. Но в отличие от случая с ремонтом автомобиля теперь становятся существенными динамические аспекты задачи. Значения уровня сахара в крови и результаты их измерения могут резко изменяться во времени, в зависимости от последнего приема пищи больным и доз инсулина, от интенсивности обмена веществ в организме, времени суток и т.д. Чтобы оценить текущее состояние больного по хронологии накопленных фактов и предсказать результаты терапевтических действий, необходимо моделировать эти изменения.

Точно такие же соображения возникают во многих других контекстах, начиная от слежения за экономической активностью определенного государства по приблизительным и частичным статистическим данным и заканчивая пониманием последовательности слов устной речи по зашумленным и неоднозначным результатам акустических измерений. Возникает вопрос: как можно промоделировать подобные динамические ситуации?

### Состояния и наблюдения

Основной подход, который будет принят в этой главе, основан на идеях, аналогичных идеям ситуационного исчисления, как описано в главе 10: процесс изменения может рассматриваться как ряд снимков, каждый из которых описывает состояние мира в данный конкретный момент времени. Каждый снимок, или **временной срез**, содержит множество случайных переменных, причем одна часть из них является наблюдаемой, а другая — нет. Для упрощения будем предполагать, что в каждом временном срезе является наблюдаемым одно и то же подмножество переменных (хотя такое требование не является строго необходимым во всем последующем изложении). Мы будем использовать  $\mathbf{x}_t$  для обозначения множества ненаблюдаемых переменных состояния во время  $t$  и  $\mathbf{e}_t$  для обозначения множества наблюдаемых переменных свидетельства. Результаты наблюдения во время  $t$  представляют собой множество  $\mathbf{E}_t = \mathbf{e}_t$ , соответствующее некоторому множеству значений  $\mathbf{e}_t$ .

Рассмотрим следующий крайне упрощенный пример. Предположим, что на каком-то секретном подземном объекте служит охранник, который никогда не выходит наружу. Он хочет знать, идет ли сегодня дождь, но единственный доступ к информации из внешнего мира охранник получает только по утрам, когда видит директора, пришедшего с зонтиком или без зонтика. Таким образом, в каждые сутки  $t$

множество  $\mathbf{E}_t$  включает единственную переменную свидетельства  $U_t$  (которая показывает, несет ли директор зонтик), а множество  $\mathbf{x}_t$  содержит единственную переменную состояния  $R_t$  (которая показывает, идет ли дождь). Другие задачи могут быть связаны с использованием более крупных множеств переменных. В частности, в примере с больным диабетом могут использоваться такие переменные свидетельства, как результаты измерения уровня сахара в крови  $MeasuredBloodSugar_t$  и частоты пульса  $PulseRate_t$ , а также переменные состояния<sup>1</sup>, такие как фактический уровень сахара в крови  $BloodSugar_t$  и содержимое желудка  $StomachContents_t$ .

Интервал между временными срезами также зависит от задачи. При контроле состояния больного диабетом подходящим интервалом может оказаться час, а не сутки. В этой главе в основном предполагается использование фиксированных конечных интервалов; это означает, что точки во времени, соответствующие временным срезам, могут быть обозначены целыми числами. Кроме того, предполагается, что последовательность состояний начинается в момент времени  $t=0$ ; к тому же по некоторым причинам, не имеющим особого значения, предполагается, что свидетельства начинают появляться в момент времени  $t=1$ , а не  $t=0$ . Таким образом, мир задачи с зонтиком может быть представлен переменными состояния  $R_0, R_1, R_2, \dots$  и переменными свидетельства  $U_1, U_2, \dots$ . Для обозначения последовательности целых чисел от  $a$  до  $b$  (включительно) будет использоваться запись  $a : b$ , а для обозначения соответствующего ряда переменных от  $\mathbf{x}_a$  до  $\mathbf{x}_b$  — запись  $\mathbf{x}_{a:b}$ . Например,  $U_{1:3}$  соответствует переменным  $U_1, U_2, U_3$ .

## Стационарные процессы и марковское предположение

После того как выбраны множества переменных состояния и переменных свидетельства для данной конкретной задачи, на следующем этапе необходимо определить зависимости между этими переменными. Мы будем следовать процедуре, описанной в главе 14, размещая переменные в некотором порядке и отвечая на вопросы об условной независимости переменных-преемников, если дано некоторое множество родительских переменных. Одним из очевидных вариантов является упорядочение переменных в их естественной временной последовательности, поскольку причина обычно предшествует результату, а мы предпочитаем вводить переменные в причинной последовательности.

Однако при таком подходе сразу же возникает одно препятствие: множество переменных является неограниченным, поскольку включает переменные состояния и переменные свидетельства для каждого временного среза. Такая ситуация фактически порождает две проблемы: во-первых, нам может потребоваться задать неограниченное количество таблиц условных вероятностей, по одной для каждой переменной в каждом срезе, во-вторых, каждая переменная может быть связана с неограниченным количеством родительских переменных.

Первая проблема разрешима на основании предположения, что изменения в состоянии мира вызваны **стационарным процессом**, т.е. процессом изменения, подчиняющимся таким законам, которые сами не изменяются во времени. (Не следует

<sup>1</sup> Обратите внимание на то, что  $BloodSugar_t$  и  $MeasuredBloodSugar_t$  представляют собой разные переменные; это позволяет отличить зашумленные результаты измерений от фактических данных.

путать *стационарные* процессы со *статическими* — в статическом процессе не изменяется само состояние.) Поэтому в мире задачи с зонтиком условная вероятность того, что директор придет с зонтиком,  $P(U_t | Parents(U_t))$ , является одинаковой для всех значений  $t$ . Таким образом, если принято предположение о стационарности, то необходимо задавать условные распределения только для переменных, относящихся к некоторому “репрезентативному” временному срезу.

Вторая проблема, касающаяся того, что приходится иметь дело с таким количеством родительских переменных, которое в принципе может стать бесконечным, решается благодаря принятию так называемого **марковского предположения**, или **предположения о марковости**, которое заключается в том, что текущее состояние зависит лишь от конечной истории предыдущих состояний. Процессы, соответствующие этому предположению, были впервые глубоко исследованы российским ученым-статистиком Андреем Марковым и называются **марковскими процессами**, или **марковскими цепями**. Существует несколько разновидностей таких процессов; простейшим из них является **марковский процесс первого порядка**, в котором текущее состояние зависит только от предыдущего состояния и не зависит от каких-либо более ранних состояний. Иными словами, состояние — это информация, которая требуется для того, чтобы сделать прогноз о будущем независимым от прошлого, если дано это состояние. С использованием принятой в этой главе системы обозначений соответствующее утверждение об условной независимости позволяет сформулировать для всех  $t$  следующее соотношение:

$$P(\mathbf{x}_t | \mathbf{x}_{0:t-1}) = P(\mathbf{x}_t | \mathbf{x}_{t-1}) \quad (15.1)$$

Таким образом, в марковском процессе первого порядка законы, описывающие, как состояние развивается во времени, полностью представлены в условном распределении  $P(\mathbf{x}_t | \mathbf{x}_{t-1})$ , которое мы будем называть **моделью перехода** для процессов первого порядка<sup>2</sup>. Моделью перехода для марковского процесса второго порядка является условное распределение  $P(\mathbf{x}_t | \mathbf{x}_{t-2}, \mathbf{x}_{t-1})$ . Структуры байесовских сетей, соответствующие марковским процессам первого и второго порядка, приведены на рис. 15.1.

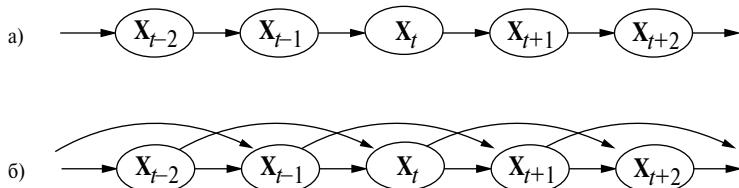


Рис. 15.1. Структуры байесовских сетей: байесовская сеть, соответствующая марковскому процессу первого порядка с состоянием, определяемым переменными  $\mathbf{x}_t$  (а); марковский процесс второго порядка (б)

Кроме ограничения количества родительских переменных, относящихся к переменным состояния  $\mathbf{x}_t$ , необходимо ограничить количество родительских переменных, относящихся к переменным свидетельства  $\mathbf{E}_t$ . Как правило, в данной главе

<sup>2</sup> Модель перехода представляет собой вероятностный аналог булевых схем обновления, описанных в главе 7, и аксиом состояния-преемника, представленных в главе 10.

предполагается, что переменные свидетельства во время  $t$  зависят только от текущего состояния:

$$\mathbf{P}(\mathbf{E}_t | \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t) \quad (15.2)$$

Условное распределение  $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$  называется **моделью восприятия** (или иногда **моделью наблюдения**), поскольку оно показывает, как фактическое состояние мира влияет на “результаты восприятия”, т.е. на переменные свидетельства. Обратите внимание на то, каково направление зависимости: “стрелки” идут от состояния к значениям результатов восприятия, поскольку именно состояние мира вынуждает результаты восприятия приобретать данные конкретные значения. Например, в мире задачи с зонтиком дождь вынуждает директора прийти с зонтиком. (Разумеется, процесс вероятностного вывода проходит в другом направлении; одним из преимуществ байесовских сетей является то, что они позволяют провести различие между направлением моделируемых зависимостей и направлением вероятностного вывода.)

Кроме модели перехода и модели восприятия, необходимо определить  $\mathbf{P}(\mathbf{x}_0)$  — распределение априорных вероятностей состояний во время 0. Эти три распределения, в сочетании с предположениями об условной независимости, приведенными в уравнениях 15.1 и 15.2, позволяют получить спецификацию полного совместного распределения по всем переменным. Для любого конечного значения  $t$  получаем следующее:

$$\mathbf{P}(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_t, \mathbf{e}_1, \dots, \mathbf{e}_t) = \mathbf{P}(\mathbf{x}_0) \prod_{i=1}^t \mathbf{P}(\mathbf{x}_i | \mathbf{x}_{i-1}) \mathbf{P}(\mathbf{e}_i | \mathbf{x}_i)$$

Предположения о независимости соответствуют очень простой структуре байесовской сети, описывающей всю систему. На рис. 15.2 показана структура сети для примера с зонтиком, включая условные распределения для модели перехода и модели восприятия.

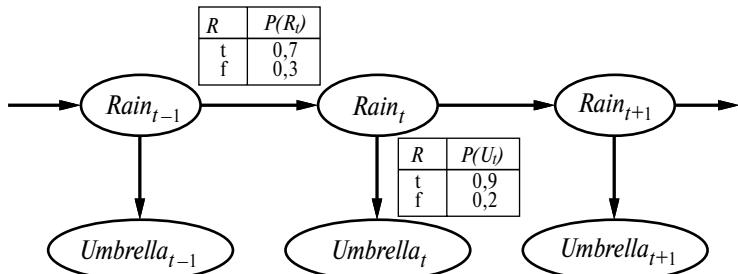


Рис. 15.2. Структура байесовской сети и распределения условных вероятностей, которые описывают мир задачи с зонтиком. Моделью перехода служит  $P(Rain_t | Rain_{t-1})$ , а моделью восприятия является  $P(Umbrella_t | Rain_t)$

В структуре, показанной на этом рисунке, предполагается использование марковского процесса первого порядка, поскольку считается, что вероятность дождя зависит только от того, был ли дождь в предыдущие сутки. Является ли такое предположение обоснованным, зависит от самой проблемной области. Марковское предположение первого порядка указывает, что переменные состояния содержат всю

информацию, необходимую для описания распределения вероятностей для следующего временного среза. Иногда такое предположение полностью соответствует истине; например, если некоторая частица совершает **случайное блуждание** вдоль оси  $x$ , изменяя свою позицию на величину  $\pm 1$  в каждый интервал времени, то применение в качестве состояния координаты  $x$  позволяет определить марковский процесс первого порядка. Иногда такое предположение является лишь приближенным, как и в случае предсказания дождя лишь на основании того, был ли дождь в предыдущие сутки. Как описано ниже, существуют два возможных метода корректировки, если такое приближение оказывается слишком неточным.

1. Повышение порядка модели марковского процесса. Например, можно перейти к использованию модели второго порядка, введя переменную  $Rain_{t-2}$  в качестве родительской по отношению к  $Rain_t$ , что позволит получить немного более точные предсказания (например, в Пало-Альто дождь очень редко идет больше двух дней подряд).
2. Расширение множества переменных состояния. Например, можно ввести переменную с обозначением времени года  $Season_t$ , чтобы иметь возможность включить хронологические данные о дождливых временах года, или ввести переменные для температуры  $Temperature_t$ , влажности  $Humidity_t$  и атмосферного давления  $Pressure_t$ , чтобы иметь возможность использовать физическую модель условий, способствующих возникновению дождя.

В упр. 15.1 предлагается показать, что первое решение (увеличение порядка модели) можно всегда переформулировать как расширение множества переменных состояния, при сохранении фиксированного порядка. Следует отметить, что введение дополнительных переменных состояния может увеличить прогностическую мощь системы, но вместе с тем повысить требования к прогнозированию, поскольку при этом придется также прогнозировать значения новых переменных. Поэтому обычно исследователи стараются найти “самодостаточное” множество переменных, а это фактически означает, что прежде всего они стремятся понять “физику” моделируемого процесса. Очевидно, что требования к точному моделированию процесса можно ослабить, если есть возможность ввести новые результаты восприятия (например, результаты измерения температуры и атмосферного давления), которые непосредственно предоставляют информацию о новых переменных состояния.

Рассмотрим, например, задачу слежения за роботом, блуждающим случайнym образом на плоскости X-Y. Можно было бы предложить, что достаточно воспользоваться таким множеством переменных состояния, как положение и скорость, — ведь для вычисления нового положения можно просто использовать законы Ньютона, а скорость будет изменяться непредсказуемо. Но если робот получает питание электрической энергией от аккумулятора, то разрядка аккумулятора будет оказывать систематическое влияние на изменение скорости. А поскольку само это влияние зависит от того, сколько электроэнергии было израсходовано во всех предыдущих маневрах, то свойство принадлежности марковской цепи к модели определенного порядка (или просто **марковость**) нарушается. Марковость цепи можно восстановить, включив уровень зарядки аккумулятора  $Battery_t$  в состав переменных состояния, которые входят в множество  $\mathbf{x}_t$ . Это позволит лучше предсказывать движения робота, но, в свою очередь, потребует создания модели для предсказания значения

$Battery_t$  на основании значения  $Battery_{t-1}$  и скорости. В некоторых случаях такое моделирование может быть выполнено достаточно надежно, а повышения точности можно будет добиться, введя для измерения уровня заряда аккумулятора новый датчик.

## 15.2. ВЕРОЯТНОСТНЫЙ ВЫВОД ВО ВРЕМЕННЫХ МОДЕЛЯХ

После определения структуры универсальной временной модели мы получаем возможность сформулировать основные задачи вероятностного вывода, которые должны быть решены с помощью этой модели, как описано ниже.

- **Фильтрация, или текущий контроль.** В этом состоит задача вычисления **доверительного состояния** — распределения апостериорных вероятностей переменных, относящихся к текущему состоянию, при наличии всех полученных к данному моменту свидетельств. Это означает, что требуется вычислить значение  $P(\mathbf{x}_t | \mathbf{e}_{1:t})$ , при условии, что фактические свидетельства поступают в виде непрерывного потока, начиная со времени  $t=1$ . В примере с зонтиком это может означать вычисление вероятности дождя сегодня, если даны все результаты наблюдений о носителе зонтика, полученные до сих пор. А *фильтрацией* называется та операция, которую должен выполнять рациональный агент в ходе слежения за текущим состоянием так, чтобы иметь возможность принимать рациональные решения (см. главу 17). Как оказалось, почти идентичные расчеты должны осуществляться в процессе определения **правдоподобия последовательности свидетельств**,  $P(\mathbf{e}_{1:t})$ .
- **Предсказание.** В этом состоит задача вычисления распределения апостериорных вероятностей значений переменных в будущем состоянии, если даны все свидетельства, полученные к данному моменту. Это означает, что может потребоваться вычислить  $P(\mathbf{x}_{t+k} | \mathbf{e}_{1:t})$  для некоторого  $k > 0$ . В примере с зонтиком такое вычисление может сводиться к определению вероятности дождя через три дня от нынешнего, если даны все результаты наблюдений за носителем зонтика, полученные до сих пор. Предсказание является полезным для оценки возможных стратегий.
- **Сглаживание, или ретроспективный анализ.** В этом состоит задача вычисления распределения апостериорных вероятностей значений переменных, относящихся к прошлому состоянию, если даны все свидетельства вплоть до нынешнего состояния. Это означает, что может потребоваться вычислить значение  $P(\mathbf{x}_k | \mathbf{e}_{1:t})$  для некоторого  $k$ , такого, что  $0 \leq k < t$ . В примере с зонтиком это может означать вероятность того, что дождь шел в прошлую среду, если даны все результаты наблюдений за носителем зонтика, полученные до сих пор. Ретроспективный анализ обеспечивает лучшую оценку информации о состоянии, которая была доступна к тому времени (поскольку включает больше свидетельств).
- **Наиболее правдоподобное объяснение.** Если дан ряд результатов наблюдений, то может потребоваться найти последовательность состояний, которые с наибольшей вероятностью стали причиной получения результатов

наблюдений. Это означает, что может потребоваться вычислить значение  $\operatorname{argmax}_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t} | \mathbf{e}_{1:t})$ . Например, если директор приходил с зонтиком в каждый из первых трех дней, а на четвертый пришел без зонтика, то наиболее вероятным объяснением становится наличие дождя в первые три дня и отсутствие в четвертый. Алгоритмы решения такой задачи являются полезными во многих приложениях, включая распознавание речи (целью которого является поиск наиболее вероятной последовательности слов, если дан ряд звуков) и реконструкцию цепочек битов, передаваемых по зашумленному каналу.

Кроме этих задач, необходимы методы для создания с помощью обучения моделей перехода и моделей восприятия на основании наблюдений. Так же как и в случае статических байесовских сетей, обучение в динамических байесовских сетях может осуществляться в виде получения побочного результата от вероятностного вывода. Вероятностный вывод предоставляет оценку того, какие переходы между состояниями фактически происходили и какие состояния привели к получению данных результатов восприятия, а эти оценки могут использоваться для обновления моделей. Обновленные модели предоставляют новые оценки, и в этом процессе происходят итерации до тех пор, пока все вычисления не сходятся. Весь этот процесс представляет собой пример применения алгоритма “ожидания–максимизации”, или **алгоритма EM** (Expectation-Maximization) (см. раздел 20.3). Важно отметить, что для обучения требуется вероятностный вывод с полным сглаживанием, а не с фильтрацией, поскольку он предоставляет лучшие оценки состояний процесса. Обучение с фильтрацией может не позволить добиться правильной сходимости; рассмотрим, например, задачу обучения раскрытию убийств: чтобы узнать из наблюдаемых переменных с помощью вероятностного вывода, что скорее всего произошло на месте убийства, всегда требуется ретроспективный анализ.

Алгоритмы для решения четырех задач вероятностного вывода, перечисленных в начале данного раздела, можно прежде всего описать на общем уровне, независимо от конкретного типа применяемой модели. Усовершенствования этих алгоритмов, относящиеся к каждой модели, будут описаны в соответствующих разделах.

## Фильтрация и предсказание

Начнем с фильтрации. Мы покажем, что эту задачу можно решить простым рекурсивным способом: если есть результаты фильтрации вплоть до момента  $t$ , можно легко вычислить результат для  $t+1$  из нового свидетельства  $\mathbf{e}_{t+1}$ . Это означает, что для некоторой функции  $f$  имеет место следующее:

$$\mathbf{P}(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{x}_t | \mathbf{e}_{1:t}))$$

Такой процесс часто называют **рекурсивной оценкой**. Соответствующее вычисление может рассматриваться как фактически состоящее из двух частей: прежде всего распределение вероятностей для текущего состояния проектируется вперед от  $t$  к  $t+1$ , затем оно обновляется с использованием нового свидетельства  $\mathbf{e}_{t+1}$ . Такое двухэтапное протекание процесса можно выразить формально весьма просто:

$$\begin{aligned} \mathbf{P}(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) && \text{(Разделение свидетельства)} \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) && \text{(Применение правила Байеса)} \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \mathbf{P}(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) && \text{(Преобразование в соответствии со свойством марковости свидетельства)} \end{aligned}$$

Здесь и далее в данной главе  $\alpha$  представляет собой нормализующую константу, используемую для того, чтобы вероятности в сумме составляли 1. Второй терм,  $P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$ , представляет одношаговое предсказание следующего состояния, а первый терм обновляет его новым свидетельством; обратите внимание на то, что значение  $P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1})$  можно получить непосредственно из модели восприятия. Теперь определим одношаговое предсказание для текущего состояния путем обусловливания вероятностей значений переменных для текущего состояния  $\mathbf{x}_t$ :

$$\begin{aligned} P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) &= \alpha P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t | \mathbf{e}_{1:t}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) \end{aligned}$$

(Преобразование в соответствии со свойством марковости) (15.3)

В операциях суммирования первым множителем является модель перехода, а вторым — распределение вероятностей для текущего состояния. Поэтому получена требуемая рекурсивная формулировка. Отфильтрованная оценка  $P(\mathbf{x}_t | \mathbf{e}_{1:t})$  может рассматриваться как “сообщение”  $\mathbf{f}_{1:t}$ , которое распространяется в прямом направлении вдоль последовательности состояний, будучи модифицируемым при каждом переходе и обновляемым при получении каждого нового результата наблюдения. Этот процесс можно представить следующим образом:

$$\mathbf{f}_{1:t+1} = \alpha \text{Forward}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1})$$

где функция `Forward` реализует обновление, описанное в уравнении 15.3.

Если все переменные состояния являются дискретными, то затраты времени на каждое обновление остаются постоянными (т.е. независимыми от  $t$ ) и потребность в пространстве также остается постоянной. (Соответствующие постоянные показатели временной и пространственной сложности, безусловно, зависят от размера пространства состояний и от конкретного типа используемой временной модели.)

*Требования ко времени и пространству для обновления должны быть постоянными, если агент с ограниченной памятью обязан следить за распределением вероятностей для текущего состояния на протяжении неограниченной последовательности наблюдений.*

Проиллюстрируем процесс фильтрации, состоящий из двух этапов, на простом примере с зонтиком (см. рис. 15.2). Предполагается, что охранник обладает с определенной степенью уверенности сведениями о распределении априорных вероятностей дождя в день 0, непосредственно до того, как началась данная последовательность наблюдений. Предположим, что данным распределением является  $P(R_0) = \langle 0.5, 0.5 \rangle$ . Теперь обработаем результаты двух наблюдений, как показано ниже.

- В день 1 директор пришел с зонтиком, поэтому  $U_1 = \text{true}$ . Результаты предсказания перехода от  $t=0$  к  $t=1$  состоят в следующем:

$$\begin{aligned} P(R_1) &= \sum_{r_0} P(R_1 | r_0) P(r_0) \\ &= \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 = \langle 0.5, 0.5 \rangle \end{aligned}$$

а их обновление с помощью свидетельства, полученного для  $t = 1$ , является таковым:

$$\begin{aligned} \mathbf{P}(R_1 | u_1) &= \alpha \mathbf{P}(u_1 | R_1) \mathbf{P}(R_1) = \alpha <0.9, 0.2><0.5, 0.5> \\ &= \alpha <0.45, 0.1> \approx <0.818, 0.182> \end{aligned}$$

- В день 2 директор также пришел с зонтиком, поэтому  $U_2 = \text{true}$ . Результаты предсказания перехода от  $t=1$  к  $t=2$  являются следующими:

$$\begin{aligned} \mathbf{P}(R_2 | u_1) &= \sum_{r_1} \mathbf{P}(R_2 | r_1) P(r_1 | u_1) \\ &= <0.7, 0.3> \times 0.818 + <0.3, 0.7> \times 0.182 \approx <0.627, 0.373> \end{aligned}$$

а обновление их с помощью свидетельства для  $t=2$  позволяет получить следующее:

$$\begin{aligned} \mathbf{P}(R_2 | u_1, u_2) &= \alpha \mathbf{P}(u_2 | R_2) \mathbf{P}(R_2 | u_1) = \alpha <0.9, 0.2><0.627, 0.373> \\ &= \alpha <0.565, 0.075> \approx <0.883, 0.117> \end{aligned}$$

Интуитивно ясно, что вероятность дождя от дня 1 ко дню 2 повышается, поскольку дождь продолжается. В упр. 15.2, а предлагается исследовать такую тенденцию дальше.

Задача **предсказания** может рассматриваться просто как фильтрация без добавления новых свидетельств. В действительности процесс фильтрации уже включает одиночное предсказание, и поэтому можно легко вывести следующую формулу рекурсивного вычисления для предсказания состояния в момент времени  $t+k+1$  на основании предсказания для  $t+k$ :

$$\mathbf{P}(\mathbf{x}_{t+k+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{x}_{t+k+1} | \mathbf{x}_{t+k}) \mathbf{P}(\mathbf{x}_{t+k} | \mathbf{e}_{1:t}) \quad (15.4)$$

Естественно, что в этом вычислении участвует только модель перехода, а не модель восприятия.

Интересно рассмотреть, что произойдет при попытке предсказывать все дальше и дальше в будущее. Как показано в упр. 15.2, б, прогнозируемое распределение для дождя сходится к фиксированной точке  $<0.5, 0.5>$ , после чего продолжает оставаться неизменным. Это — так называемое **стационарное распределение** для марковского процесса, определяемого с помощью модели перехода (см. также с. 692). О свойствах таких распределений и о ~~и~~ продолжительности смешивания (грубо говоря, о затратах времени, необходимых для достижения фиксированной точки) известно очень многое. С точки зрения практики эти знания сводятся к печальному выводу, что любая попытка предсказать фактическое состояние для количества этапов, составляющего не больше чем небольшую часть количества, соответствующего продолжительности смешивания, обречена на неудачу. Чем более неопределенной является модель перехода, тем короче будет продолжительность смешивания и тем более туманным становится будущее.

Рекурсия в прямом направлении может использоваться не только для фильтрации и предсказания, но и для вычисления **правдоподобия** последовательности свидетельств,  $P(\mathbf{e}_{1:t})$ . Такое значение может оказаться применимым, если потребуется сравнить различные временные модели, которые способны вырабатывать одну и ту

же последовательность свидетельств; например, в разделе 15.6 сравниваются различные слова, при произношении которых может создаваться одна и та же последовательность звуков. Для такой рекурсии используется сообщение о правдоподобии  $\ell_{1:t} = P(\mathbf{x}_t | \mathbf{e}_{1:t})$ . Несложно показать, что справедливо следующее соотношение:

$$\ell_{1:t+1} = \text{Forward}(\ell_{1:t}, \mathbf{e}_{t+1})$$

После вычисления  $\ell_{1:t}$  получим фактическое значение правдоподобия, исключая путем суммирования значение  $\mathbf{x}_t$ :

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \ell_{1:t}(\mathbf{x}_t) \quad (15.5)$$

## Сглаживание

Как было указано выше, **сглаживание** — это процесс вычисления распределения вероятностей значений переменных в прошлых состояниях при наличии свидетельств вплоть до нынешнего состояния, иными словами,  $P(\mathbf{x}_k | \mathbf{e}_{1:t})$  для  $1 \leq k < t$  (рис. 15.3.) Такие вычисления наиболее удобно разбить на две части: применительно к свидетельствам вплоть до момента времени  $k$  и к свидетельствам от  $k+1$  до  $t$ :

$$\begin{aligned} P(\mathbf{x}_k | \mathbf{e}_{1:t}) &= P(\mathbf{x}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha P(\mathbf{x}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{x}_k, \mathbf{e}_{1:k}) \quad (\text{Применение правила Байеса}) \\ &= \alpha P(\mathbf{x}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{x}_k) \quad (\text{Применение правила условной независимости}) \\ &= \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t} \end{aligned} \quad (15.6)$$

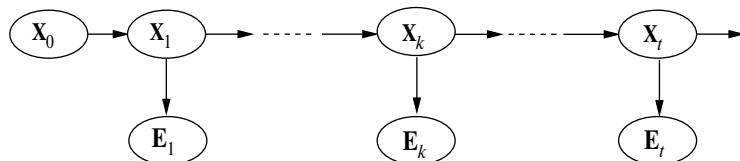
где используется сообщение  $\mathbf{b}_{k+1:t} = P(\mathbf{e}_{k+1:t} | \mathbf{x}_k)$ , определяемое как “обратное”, по аналогии с прямым сообщением  $\mathbf{f}_{1:k}$ . Прямое сообщение  $\mathbf{f}_{1:k}$  может быть вычислено путем фильтрации в прямом направлении от 1 до  $k$  в соответствии с уравнением 15.3. Как оказалось, обратное сообщение  $\mathbf{b}_{k+1:t}$  может быть вычислено с помощью рекурсивного процесса, который осуществляется в обратном направлении от  $t$ :

$$\begin{aligned} P(\mathbf{e}_{k+1:t} | \mathbf{x}_k) &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_k, \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k) \quad (\text{Обусловливание по } \mathbf{x}_{k+1}) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k) \\ &\quad (\text{Применение правила условной независимости}) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k) \end{aligned} \quad (15.7)$$

где последний этап преобразования следует из свойства условной независимости  $\mathbf{e}_{k+1}$  и  $\mathbf{e}_{k+2:t}$ , если дано  $\mathbf{x}_{k+1}$ . Из трех множителей в этой операции суммирования первый и третий можно получить непосредственно с помощью модели, а второй представляет собой “рекурсивный вызов”. С использованием обозначения для сообщений получим следующее:

$$\mathbf{b}_{k+1:t} = \text{Backward}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$$

где функция `Backward` осуществляет обновление, описанное в уравнении 15.7. Как и при рекурсии в прямом направлении, время и пространство, требуемые для каждого обновления, являются постоянными и поэтому независимыми от  $t$ .



*Рис. 15.3. Процесс сглаживания, в котором вычисляется  $P(\mathbf{x}_k | \mathbf{e}_{1:t})$ , распределение апостериорных вероятностей значений переменных в состоянии, наблюдавшемся в какой-то прошлый момент времени  $k$ , если дана полная последовательность наблюдений от 1 до  $t$*

Теперь становится вполне очевидно, что оба терма, приведенные в уравнении 15.6, можно вычислить с помощью рекурсий по времени, одна из которых проходит в прямом направлении от 1 до  $k$  и в которой используется уравнение фильтрации 15.3, а другая проходит в обратном направлении от  $t$  до  $k+1$  и в ней используется уравнение 15.7. Следует отметить, что этап обратной рекурсии инициализируется с помощью выражения  $\mathbf{b}_{t+1:t} = P(\mathbf{e}_{t+1:t} | \mathbf{x}_t) = \mathbf{1}$ , где  $\mathbf{1}$  — вектор из единиц (поскольку  $\mathbf{e}_{t+1:t}$  — пустая последовательность, вероятность наблюдения равна 1).

Итак, применим этот алгоритм к примеру с зоником и вычислим сглаженную оценку вероятности дождя в момент времени  $t=1$  по данным наблюдений за появлением директора с зоником в дни 1 и 2. Согласно уравнению 15.6, это значение определяется следующим образом:

$$P(R_1 | u_1, u_2) = \alpha P(R_1 | u_1) P(u_2 | R_1) \quad (15.8)$$

Как нам уже известно, первый терм равен  $<0.818, 0.182>$ , согласно результатам применения процесса прямой фильтрации, описанного выше. Второй терм можно вычислить, применив обратную рекурсию по уравнению 15.7:

$$\begin{aligned} P(u_2 | R_1) &= \sum_{r_2} P(u_2 | r_2) P(r_2 | R_1) \\ &= (0.9 \times 1 \times <0.7, 0.3>) + (0.2 \times 1 \times <0.3, 0.7>) = \\ &= <0.69, 0.41> \end{aligned}$$

Подставляя эти значения в уравнение 15.8, можно найти, что сглаженная оценка вероятности дождя в день 1 такова:

$$P(R_1 | u_1, u_2) = \alpha <0.818, 0.182> \times <0.69, 0.41> \approx <0.883, 0.117>$$

Таким образом, в данном случае сглаженная оценка выше, чем отфильтрованная оценка (0.818). Это связано с тем, что появление директора с зонтиком в день 2 повышает вероятность того, что в день 2 шел дождь; в свою очередь, поскольку дожди обычно являются затяжными, это приводит к повышению вероятности дождя и в день 1.

И для прямой, и для обратной рекурсии требуется постоянное количество времени в расчете на каждый этап; поэтому временная сложность сглаживания по отношению к свидетельству  $e_{1:t}$  составляет  $O(t)$ . Этим выражением определяется также сложность сглаживания в конкретном интервале времени  $k$ . А если требуется пройти сглаживание всей последовательности, то один из очевидных методов состоит в выполнении всего процесса сглаживания по одному разу для каждого интервала времени, в котором должно быть выполнено сглаживание. Такой подход приводит к получению временной сложности  $O(t^2)$ . Но в лучшем подходе можно было бы использовать очень простое приложение динамического программирования, чтобы свести сложность к величине  $O(t)$ . Один из намеков на то, как это сделать, можно найти в приведенном выше анализе примера с зонтиком, где была показана возможность повторного использования результатов прямой фильтрации. Ключом к созданию алгоритма с линейными затратами времени является регистрация результатов прямой фильтрации по всей последовательности. В таком случае можно выполнить обратную рекурсию от  $t$  вплоть до 1, вычисляя сглаженную оценку для каждого этапа  $k$  из вычисленного обратного сообщения  $b_{k+1:t}$  и хранимого прямого сообщения  $f_{1:k}$ . Этот алгоритм, обоснованно названный **прямым–обратным алгоритмом** (forward-backward algorithm), показан в листинге 15.1.

**Листинг 15.1. Прямой–обратный алгоритм для вычисления апостериорных вероятностей последовательности состояний при наличии последовательности наблюдений. Операторы `Forward` и `Backward` определены в уравнениях 15.3 и 15.7**

---

```

function Forward-Backward(ev, prior) returns вектор распределений
вероятностей
    inputs: ev, вектор значений свидетельств для этапов 1,...,t
            prior, распределение априорных вероятностей в начальном
            состоянии,  $P(\mathbf{X}_0)$ 
    local variables: fv, вектор прямых сообщений для этапов 0,...,t
            b, представление обратного сообщения,
            первоначально состоящее из одних единиц, 1
            sv, вектор сглаженных оценок для этапов 1,...,t

    fv[0]  $\leftarrow$  prior
    for i=1 to t do
        fv[i]  $\leftarrow$  Forward(fv[i-1], ev[i])
    for i=t downto 1 do
        sv[i]  $\leftarrow$  Normalize(fv[i]  $\times$  b)
        b  $\leftarrow$  Backward(b, ev[i])
    return sv

```

---

Внимательный читатель должен был заметить, что структура байесовской сети, показанной на рис. 15.3, представляет собой **полидерево**, согласно терминологии главы 14. Это означает, что непосредственное применение алгоритма кластеризации также приводит к созданию алгоритма с линейными затратами времени, который

вычисляет сглаженные оценки для всей последовательности. Итак, вполне понятно, что прямой–обратный алгоритм фактически представляет собой частный случай алгоритма распространения в полидереве, используемого в сочетании с методами кластеризации (хотя эти два алгоритма были разработаны независимо).

Прямой–обратный алгоритм лежит в основе вычислительных методов, применяемых во многих таких приложениях, где приходится иметь дело с последовательностями зашумленных результатов наблюдений, начиная от распознавания речи и заканчивая слежением за самолетами с помощью радара. Как было описано выше, с точки зрения практики он имеет два недостатка. Первым из них является пространственная сложность, которая может оказаться слишком высокой применительно к приложениям, где пространство состояний велико, а последовательности имеют большую длину. В нем используется пространство  $O(|\mathbf{f}|t)$ , где  $|\mathbf{f}|$  — размер представления прямого сообщения. Такую потребность в пространстве можно уменьшить до  $O(|\mathbf{f}|\log t)$  с помощью соответствующего увеличения временной сложности на коэффициент  $\log t$ , как показано в упр. 15.3. В некоторых случаях (см. раздел 15.3) может использоваться алгоритм с постоянными потребностями в пространстве, но без лишних затрат времени.

Вторым недостатком этого несложного алгоритма является то, что он требует модификации для использования в оперативных приложениях, где сглаженные оценки должны вычисляться для более ранних временных срезов по мере того, как к концу последовательности непрерывно добавляются новые наблюдения. Наиболее часто возникает требование по обеспечению **сглаживания с постоянным запаздыванием**, при котором необходимо вычислять сглаженную оценку  $\mathbf{P}(\mathbf{x}_{t-d} | \mathbf{e}_{1:t})$  для постоянного значения  $d$ . Это означает, что сглаживание выполняется для временного среза, отстоящего на  $d$  этапов от текущего времени  $t$ ; по мере возрастания  $t$  сглаживание не должно отставать. Безусловно, что прямой–обратный алгоритм можно вызывать на выполнение применительно к данным  $d$ -этапного “окна” по мере добавления результатов каждого нового наблюдения, но такой подход, по-видимому, является неэффективным. В разделе 15.3 будет показано, что сглаживание с постоянным запаздыванием в некоторых случаях может выполняться за постоянное время в расчете на каждое обновление, независимо от запаздывания  $d$ .

### Поиск наиболее вероятной последовательности

Предположим, что `[true, true, false, true, true]` — последовательность истинностных значений с результатами наблюдений за появлением директора с зонтиком, которые проводил охранник в течение первых пяти дней своей работы. Какая последовательность состояний погоды может стать наиболее вероятным объяснением для этих данных? Означает ли отсутствие зонтика в день 3, что не было дождя или директор забыл его взять? А если в день 3 не было дождя, то, возможно, дождь не шел и в день 4 (поскольку погода обычно является устойчивой), однако директор захватил с собой зонтик просто на всякий случай. В целом существуют  $2^5$  возможных последовательностей состояний погоды, которые могут быть приняты в качестве объяснения. Но есть ли способ найти наиболее вероятную из этих последовательностей, не перебирая их все?

Один из подходов, который может быть опробован, состоит в использовании следующей процедуры с линейными затратами времени: с помощью алгоритма

сглаживания найти распределения апостериорных вероятностей погоды на каждом временном интервале; после этого составить последовательность, используя на каждом этапе наиболее вероятное состояние погоды, соответствующее этим апостериорным вероятностям. Но такой подход должен вызвать у читателя сомнения, поскольку апостериорные вероятности, вычисленные с помощью сглаживания, представляют собой распределения вероятностей для отдельных временных интервалов, тогда как, чтобы найти наиболее вероятную последовательность, необходимо рассматривать совместные вероятности по всем временным интервалам. Соответствующие результаты фактически могут оказаться довольно различными (см. упр. 15.4).

Тем не менее алгоритм поиска наиболее вероятной последовательности с линейными затратами времени существует, но чтобы его найти, необходимо продумать все обстоятельства немного более тщательно. Алгоритм должен быть основан на том же свойстве марковости (неизменности порядка марковской цепи), которое позволило создать эффективные алгоритмы фильтрации и сглаживания. Проще всего можно подойти к решению этой задачи, рассматривая каждую последовательность как путь в графе, узлами которого являются возможные состояния на каждом временном интервале. Такой график показан для мира задачи с зонтиком на рис. 15.4, а. Теперь рассмотрим задачу поиска наиболее вероятного пути через этот график, где значение правдоподобия любого пути представляет собой произведение вероятностей перехода вдоль этого пути и вероятностей конкретных наблюдений в каждом состоянии. В частности, сосредоточимся на тех путях, которые достигают состояния  $Rain_5 = true$ . Ввиду наличия свойства марковости можно прийти к заключению, что наиболее вероятный путь к состоянию  $Rain_5 = true$  состоит из наиболее вероятного пути к некоторому состоянию, достигнутому во время 4, за которым следует переход в состояние  $Rain_5 = true$ , а состоянием во время 4, которое станет частью пути к состоянию  $Rain_5 = true$ , является именно то состояние, которое максимизирует значение правдоподобия этого пути. Иными словами, существует рекурсивная связь между наиболее вероятными путями в каждое состояние  $\mathbf{x}_{t+1}$  и наиболее вероятными путями в каждое состояние  $\mathbf{x}_t$ . Эту связь можно записать в виде уравнения, соединяющего вероятности путей, следующим образом:

$$\begin{aligned} & \max_{\mathbf{x}_1 \dots \mathbf{x}_t} P(\mathbf{x}_1, \dots, \mathbf{x}_t, \mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) \\ &= \alpha P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \max_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) \max_{\mathbf{x}_1 \dots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{e}_{1:t}) \end{aligned} \quad (15.9)$$

Уравнение 15.9 идентично уравнению фильтрации 15.3, за исключением описанных ниже отличий.

1. Прямое сообщение  $f_{1:t} = P(\mathbf{x}_t \mid \mathbf{e}_{1:t})$  заменяется следующим сообщением, т.е. вероятностями наиболее правдоподобного пути в каждое состояние  $\mathbf{x}_t$ :
 
$$m_{1:t} = \max_{\mathbf{x}_1 \dots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{e}_{1:t})$$
2. Суммирование по переменным  $\mathbf{x}_t$  в уравнении 15.3 заменяется максимизацией по переменным  $\mathbf{x}_t$  в уравнении 15.9.

Таким образом, алгоритм вычисления наиболее вероятной последовательности аналогичен алгоритму фильтрации: он проходит в прямом направлении вдоль по-

следовательности, вычисляя сообщение  $m$  на каждом временном интервале с помощью уравнения 15.9. Ход этих вычислений показан на рис. 15.4, б). В конечном итоге этот алгоритм позволяет получить вероятность наиболее правдоподобной последовательности, достигающей каждого из заключительных состояний. Поэтому с его помощью можно легко выбрать последовательность, которая является наиболее правдоподобной среди всех прочих (соответствующие ей состояния обозначены на рисунке жирными контурами). Для того чтобы иметь возможность определить фактически наблюданную последовательность, а не просто вычислить ее вероятность, в этом алгоритме необходимо также сопровождать указатели от каждого состояния обратно к наилучшему состоянию, приведшему к нему (эти указатели показаны на рисунке жирными стрелками); соответствующую последовательность можно найти, проследовав по указателям в обратном направлении от наилучшего заключительного состояния.

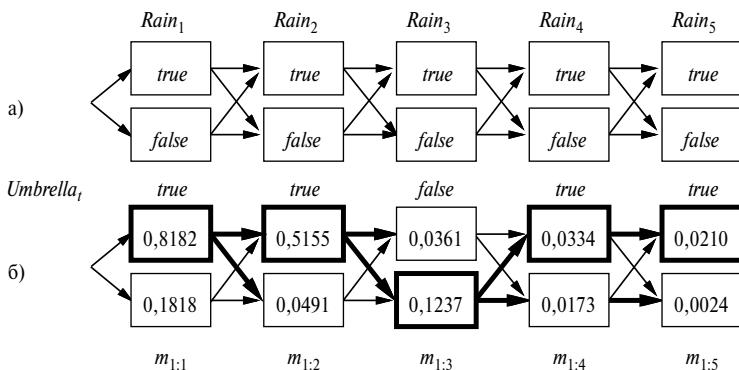


Рис. 15.4. Пример применения алгоритма Витерби: возможные последовательности состояний для переменной  $Rain_t$  могут рассматриваться как пути через граф возможных состояний на каждом временном интервале (состояния обозначены прямоугольниками для предотвращения путаницы с узлами в байесовской сети) (а); результаты применения алгоритма Витерби к последовательности наблюдений за появлением директора с зонтиком,  $[true, true, false, true, true]$  (б). Для каждого значения  $t$  показано значение сообщения  $m_{1:t}$ , которое представляет собой вероятность наилучшей последовательности, достигающей каждого состояния во время  $t$ . Кроме того, для каждого состояния ведущая в него жирная стрелка показывает его наилучшего преемника, согласно оценке, представляющей собой произведение вероятности предшествующей последовательности и вероятности перехода. Наиболее вероятную последовательность можно определить, проходя по жирным стрелкам в обратном направлении от наилучшего временного интервала времени  $m_{1:5}$ .

Только что описанный алгоритм называется **алгоритмом Витерби** в честь его разработчика. Временная сложность этого алгоритма, как и алгоритма фильтрации, линейно зависит от  $t$ , от длины последовательности. Но в отличие от алгоритма фильтрации его потребность в пространстве также линейно зависит от  $t$ . Это связано с тем, что в алгоритме Витерби необходимо следить за указателями, которые обозначают наилучшую последовательность, ведущую к каждому состоянию.

### 15.3. СКРЫТЫЕ МАРКОВСКИЕ МОДЕЛИ

В предыдущем разделе были разработаны алгоритмы формирования временных вероятностных рассуждений с использованием общей инфраструктуры, которая была независимой от конкретной формы моделей перехода и моделей восприятия. В этом и следующих двух разделах будут описаны более конкретные модели и приложения, которые иллюстрируют мощь этих простых алгоритмов, а в некоторых случаях допускают дополнительные усовершенствования.

Начнем с описания скрытой марковской модели, или сокращенно **HMM** (Hidden Markov Model). Любая модель HMM — это времененная вероятностная модель, в которой состояние процесса описано с помощью единственной дискретной случайной переменной. Возможными значениями этой переменной являются возможные состояния мира. Поэтому пример с зонтиком, описанный в предыдущем разделе, представляет собой одну из моделей HMM, поскольку в нем применяется лишь единственная переменная состояния,  $Rain_t$ . Во временную модель могут быть введены дополнительные переменные состояния без выхода за пределы инфраструктуры HMM, но только путем комбинирования всех переменных состояния в единственную “мегапеременную”, значениями которой являются все возможные кортежи значений отдельных переменных состояния. В данном разделе будет показано, что благодаря такой жестко регламентированной структуре моделей HMM появляется возможность создавать очень простые и элегантные матричные реализации всех основных алгоритмов<sup>3</sup>. В разделе 15.6 показано, как модели HMM используются для распознавания речи.

#### Упрощенные матричные алгоритмы

При использовании единственной, дискретной переменной состояния  $X_t$  можно определить более сжатую форму представлений модели перехода, модели восприятия, а также прямых и обратных сообщений. Предположим, что переменная состояния  $X_t$  имеет значения, обозначенные целыми числами  $1, \dots, S$ , где  $S$  — количество возможных состояний. В таком случае модель перехода  $P(X_t | X_{t-1})$  преобразуется в матрицу  $\mathbf{T}$  с размерами  $S \times S$ , где

$$\mathbf{T}_{ij} = P(X_t=j | X_{t-1}=i)$$

Таким образом,  $\mathbf{T}_{ij}$  представляет собой вероятность перехода из состояния  $i$  в состояние  $j$ . Например, матрица перехода для мира задачи с зонтиком выглядит следующим образом:

$$\mathbf{T} = P(X_t | X_{t-1}) = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

Переведем также в матричную форму модель восприятия. В данном случае, поскольку известно, что значение переменной свидетельства  $E_t$  равно, скажем,  $e_t$ , необходимо использовать только ту часть модели, в которой определена вероятность появления значения  $e_t$ . Для каждого временного интервала  $t$  мы составим диагональную матрицу  $\mathbf{O}_t$ , диагональные элементы которой задаются значениями

<sup>3</sup> Читателю, не знакомому с основными операциями над векторами и матрицами, может потребоваться обратиться к приложению А, прежде чем продолжать чтение данного раздела.

$P(e_t | X_t = i)$ , а остальные элементы равны 0. Например, в мире задачи с зонтиком в день 1 было получено значение  $U_1 = \text{true}$ , поэтому, согласно рис. 15.2, имеет место следующее:

$$\mathbf{o}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$$

Теперь, если для представления прямых и обратных сообщений будут использоваться векторы столбцов, то все вычисления преобразуются в простые матрично-векторные операции. Прямое уравнение 15.3 принимает следующий вид:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \quad (15.10)$$

а обратное уравнение 15.7 становится следующим:

$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t} \quad (15.11)$$

Как показывают эти уравнения, временная сложность прямого–обратного алгоритма (см. листинг 15.1), применяемого к последовательности с длиной  $t$ , равна  $O(S^2 t)$ , поскольку на каждом этапе требуется умножать вектор с  $S$  элементами на матрицу  $S \times S$ . Потребность в пространстве измеряется величиной  $O(St)$ , так как при проходе в прямом направлении необходимо сохранить в памяти  $t$  векторов с размером  $S$ .

Такая матричная формулировка не только становится изящным описанием алгоритмов фильтрации и сглаживания для моделей НММ, но и открывает возможности для создания улучшенных алгоритмов. Первым из таких алгоритмов является простой вариант прямого–обратного алгоритма, который обеспечивает выполнение сглаживания за счет использования постоянного пространства, независимо от длины последовательности. Идея этого алгоритма состоит в том, что для выполнения сглаживания в любом конкретном временном срезе  $k$  требуется одновременное присутствие в памяти и прямого, и обратного сообщений,  $\mathbf{f}_{1:k}$  и  $\mathbf{b}_{k+1:t}$ , согласно уравнению 15.6. В прямом–обратном алгоритме это достигается за счет хранения значений  $\mathbf{f}$ , вычисленных во время прохода в прямом направлении, для того, чтобы они были доступными во время прохода в обратном направлении. Еще один способ достижения этой цели состоит в использовании одного прохода, в котором и значение  $\mathbf{f}$ , и значение  $\mathbf{b}$  распространяются в одном и том же направлении. Например, можно добиться распространения “прямого” сообщения  $\mathbf{f}$  в обратном направлении, преобразовав уравнение 15.10 таким образом, чтобы оно действовало в другом направлении, как показано ниже.

$$\mathbf{f}_{1:t} = \alpha' (\mathbf{T}^\top)^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1}$$

Этот модифицированный алгоритм сглаживания действует так, что в нем вначале осуществляется стандартный проход в прямом направлении для вычисления значения  $\mathbf{f}_{t:t}$  (при этом все промежуточные результаты уничтожаются), после чего выполняется проход в обратном направлении для совместного вычисления и  $\mathbf{b}$ , и  $\mathbf{f}$ , а затем эти значения используются для вычисления сглаженной оценки для каждого интервала. Поскольку требуется только одна копия каждого сообщения, потребности в памяти являются постоянными (т.е. независимыми от  $t$ , длины последовательности). Тем не менее этот алгоритм имеет одно существенное ограничение — в нем требуется, чтобы матрица перехода была обратимой, а модель восприятия не имела нулей, иными словами, чтобы каждое наблюдение было возможным в любом состоянии.

Матричная формулировка открывает возможности усовершенствования еще в одном направлении — в области создания методов оперативного сглаживания с постоянным запаздыванием. Тот факт, что сглаживание может быть выполнено за счет постоянных затрат пространства, наводит на мысль, что может существовать эффективный рекурсивный алгоритм для оперативного сглаживания, т.е. алгоритм, временная сложность которого остается независимой от величины запаздывания. Предположим, что запаздывание равно  $d$ ; это означает, что сглаживание проводится во временном срезе  $t-d$ , притом что текущее время равно  $t$ . Согласно уравнению 15.6, для среза  $t-d$  необходимо рассчитать значение следующего выражения:

$$\alpha \mathbf{f}_{1:t-d} \mathbf{b}_{t-d+1:t}$$

В таком случае после поступления новых результатов наблюдения необходимо будет вычислить следующее выражение для среза  $t-d+1$ :

$$\alpha \mathbf{f}_{1:t-d+1} \mathbf{b}_{t-d+2:t+1}$$

Как можно выполнить эту операцию инкрементно? Прежде всего, можно вычислить  $\mathbf{f}_{1:t-d+1}$  из  $\mathbf{f}_{1:t-d}$  с использованием стандартного процесса фильтрации, в соответствии с уравнением 15.3.

Инкрементное вычисление обратного сообщения является более сложным, поскольку не существует простого соотношения между старым обратным сообщением  $\mathbf{b}_{t-d+1:t}$  и новым обратным сообщением  $\mathbf{b}_{t-d+2:t+1}$ . Вместо этого рассмотрим соотношение между старым обратным сообщением  $\mathbf{b}_{t-d+1:t}$  и обратным сообщением в начале последовательности,  $\mathbf{b}_{t+1:t}$ . Для этого  $d$  раз применим уравнение 15.11, чтобы получить следующее уравнение:

$$\mathbf{b}_{t-d+1:t} = \left( \prod_{i=t-d+1}^t \mathbf{TO}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t} \mathbf{1} \quad (15.12)$$

где матрица  $\mathbf{B}_{t-d+1:t}$  является произведением последовательности матриц  $\mathbf{O}$  и  $\mathbf{T}$ . Матрицу  $\mathbf{B}$  можно рассматривать как “оператор преобразования”, который преобразует более позднее обратное сообщение в более раннее. Аналогичное уравнение остается справедливым для новых обратных сообщений, сформированных после поступления результатов следующего наблюдения:

$$\mathbf{b}_{t-d+2:t+1} = \left( \prod_{i=t-d+2}^{t+1} \mathbf{TO}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1} \mathbf{1} \quad (15.13)$$

Исследуя выражения для произведений в уравнениях 15.12 и 15.13, можно определить, что их связывает между собой простое соотношение, — чтобы получить второе произведение, достаточно “разделить” первое произведение на первый элемент  $\mathbf{TO}_{t-d+1}$  и умножить на новый последний элемент  $\mathbf{TO}_{t+1}$ . Поэтому на языке алгебры матриц можно представить следующее простое соотношение между старой и новой матрицами  $\mathbf{B}$ :

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1} \mathbf{T}^{-1} \mathbf{B}_{t-d+1:t} \mathbf{TO}_{t+1} \quad (15.14)$$

Это уравнение предоставляет способ вычисления инкрементного обновления для матрицы  $\mathbf{B}$ , которая, в свою очередь (в силу уравнения 15.13), позволяет вычислить новое обратное сообщение  $\mathbf{b}_{t-d+2:t+1}$ . Полный алгоритм, в котором предусматривается хранение и обновление значений  $\mathbf{f}$  и  $\mathbf{B}$ , показан в листинге 15.2.

**Листинг 15.2. Алгоритм сглаживания с постоянным временным запаздыванием на  $d$  этапов, реализованный как оперативный алгоритм, который выводит новую сглаженную оценку после получения данных наблюдения, относящихся к новому временному интервалу**

---

```

function Fixed-Lag-Smoothing( $e_t$ ,  $hmm$ ,  $d$ ) returns распределение по  $\mathbf{x}_{t-d}$ 
  inputs:  $e_t$ , текущее свидетельство для временного интервала  $t$ 
            $hmm$ , скрытая марковская модель с матрицей переходов  $\mathbf{T}$ 
           с размерами  $S \times S$ 
            $d$ , величина запаздывания при сглаживании
  static:  $t$ , текущее время, первоначально равно 1
            $\mathbf{f}$ , распределение вероятностей (прямое сообщение  $\mathbf{P}(X_t | e_{1:t})$ ),
           первоначально  $Prior[hmm]$ 
            $\mathbf{B}$ , матрица  $d$ -этапного обратного преобразования,
           первоначально матрица идентичного преобразования
            $e_{t-d:t}$ , двухсторонний список свидетельств от  $t-d$  до  $t$ ,
           первоначально пустой
  local variables:  $\mathbf{O}_{t-d}, \mathbf{O}_t$ , диагональные матрицы, содержащие
           информацию модели восприятия

  добавить  $e_t$  к концу списка  $e_{t-d:t}$ 
   $\mathbf{O}_t \leftarrow$  диагональная матрица, содержащая  $\mathbf{P}(e_t | X_t)$ 
  if  $t > d$  then
     $\mathbf{f} \leftarrow \text{Forward}(\mathbf{f}, e_t)$ 
    удалить  $e_{t-d-1}$  с начала списка  $e_{t-d:t}$ 
     $\mathbf{O}_{t-d} \leftarrow$  диагональная матрица, содержащая  $\mathbf{P}(e_{t-d} | X_{t-d})$ 
     $\mathbf{B} \leftarrow \mathbf{O}_{t-d}^{-1} \mathbf{T}^{-1} \mathbf{B} \mathbf{O}_t$ 
  else  $\mathbf{B} \leftarrow \mathbf{B} \mathbf{O}_t$ 
   $t \leftarrow t + 1$ 
  if  $t > d$  then return Normalize( $\mathbf{f} \times \mathbf{B}^{-1}$ ) else return пустое значение

```

---

## 15.4. ФИЛЬТРЫ КАЛМАНА

Представьте себе, что вы внимательно следите за маленькой птицей, летящей через плотную листву джунглей в сумраке: вы замечаете краткие промежуточные этапы движения и пытаешься угадать, где сейчас находится птица и где она появится в следующий раз, чтобы ее не потерять. Или вообразите себя на месте оператора радара во Второй мировой войне, который напряженно вглядывается в крошечный, блуждающий блик, появляющийся на экране через каждые 10 секунд. А если вернуться в прошлое еще дальше, вообразите, что вы, как Кеплер, пытаетесь реконструировать орбиты движения планет из совокупности крайне неточных угловых измерений, полученных через нерегулярные и неточно измеренные интервалы времени. Во всех этих случаях осуществляется попытка оценить показатели состояния физической системы (например, положение и скорость) на основе зашумленных результатов на-

блодений во времени. Эту задачу можно сформулировать как вероятностный логический вывод во временной вероятностной модели, где модель перехода описывает физические основы движения, а модель восприятия описывает процесс измерения. В данном разделе рассматриваются специальные формы представления и алгоритмы вероятностного вывода, которые были разработаны для решения задач такого рода; описанный здесь метод называется **калмановской фильтрацией** в честь его разработчика Рудольфа Э. Калмана.

Очевидно, что для определения состояния этой системы требуется несколько непрерывных переменных. Например, траектория полета птицы может быть задана с помощью данных о положении ( $X, Y, Z$ ) и скорости ( $\dot{X}, \dot{Y}, \dot{Z}$ ) в каждый момент времени. Необходимо также иметь соответствующие плотности условных вероятностей для представления модели перехода и модели восприятия; как и в главе 14, в данной главе будут использоваться **линейные гауссовые распределения**. Это означает, что следующее состояние  $\mathbf{x}_{t+1}$  должно представлять собой линейную функцию от текущего состояния  $\mathbf{x}_t$  с добавлением какого-то гауссова шума, а такое предположение, как оказалось, является весьма оправданным на практике. Рассмотрим, например, координату  $X$  птицы, игнорируя на данный момент все другие координаты. Допустим, что интервал между наблюдениями равен  $\Delta$ , и предположим, что птица летит с постоянной скоростью; в таком случае данные об обновлении ее положения определяются с помощью следующего уравнения:

$$x_{t+\Delta} = x_t + \dot{x} \Delta$$

Если мы введем в него гауссов шум, то получим линейную гауссову модель перехода:

$$P(x_{t+\Delta} = x_{t+\Delta} | X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t \Delta, \sigma) (x_{t+\Delta})$$

Структура байесовской сети для системы с переменными, описывающими положение  $\mathbf{x}_t$  и скорость  $\dot{\mathbf{x}}_t$ , показана на рис. 15.5. Обратите внимание на то, что это — весьма специфичная форма линейной гауссовой модели; общая форма этой модели будет описана ниже в этом разделе; она охватывает колоссальный спектр приложений, выходящий за пределы простых примеров моделирования движений, приведенных в первом абзаце данного раздела. Читателю может потребоваться обратиться к приложению А для ознакомления с некоторыми математическими свойствами гауссовых распределений; для наших непосредственных целей наиболее важным из них является то, что **многомерное гауссово** распределение для  $d$  переменных задается  $d$ -элементным средним  $\mu$  и матрицей ковариации  $\Sigma$  с размерами  $d \times d$ .

## Обновление гауссовых распределений

В главе 14 было описано ключевое свойство семейства линейных гауссовых распределений: при стандартных операциях в байесовской сети оно остается замкнутым. В этом разделе данное утверждение будет уточнено в контексте фильтрации с помощью временной вероятностной модели. Ниже перечислены требуемые свойства, соответствующие процессу двухэтапного вычисления результатов фильтрации с помощью уравнения 15.3.

1. Если текущее распределение  $P(\mathbf{x}_t | \mathbf{e}_{1:t})$  является гауссовым, а модель перехода  $P(\mathbf{x}_{t+1} | \mathbf{x}_t)$  — линейной гауссовой, то распределение, прогнозируемое

на один этап вперед, которое задается с помощью следующего уравнения, также представляет собой гауссово распределение:

$$P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) d\mathbf{x}_t \quad (15.15)$$

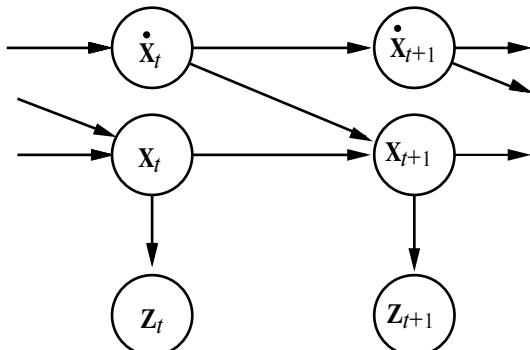


Рис. 15.5. Структура байесовской сети для линейной динамической системы с переменными, определяющими положение  $\mathbf{x}_t$ , скорость  $\dot{\mathbf{x}}_t$  и результаты измерения позиции  $\mathbf{z}_t$

- Если прогнозируемое распределение  $P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$  является гауссовым, а модель восприятия  $P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1})$  — линейной гауссовой, то после обусловлиивания вероятности на основании нового свидетельства следующее обновленное распределение также является гауссовым:

$$P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) \quad (15.16)$$

Таким образом, оператор Forward для калмановской фильтрации принимает на входе гауссово прямое сообщение  $\mathbf{f}_{1:t}$ , заданное с помощью среднего  $\mu_t$  и матрицы ковариации  $\Sigma_t$ , и вырабатывает новое многомерное гауссово прямое сообщение  $\mathbf{f}_{1:t+1}$ , заданное с помощью среднего  $\mu_{t+1}$  и матрицы ковариации  $\Sigma_{t+1}$ . Итак, начиная с гауссова априорного сообщения  $\mathbf{f}_{1:0}=P(\mathbf{x}_0)=N(\mu_0, \Sigma_0)$  и проводя фильтрацию с помощью линейной гауссовой модели, мы можем получить гауссово распределение вероятностей состояний для любых временных срезов.

Очевидно, что этот научный результат является привлекательным и изящным, но почему он имеет такое важное значение? Причина этого состоит в том, что за исключением нескольких частных случаев, подобных рассматриваемому, ~~в процессе фильтрации с помощью непрерывных или гибридных (как дискретных, так и непрерывных) сетей вырабатываются распределения вероятностей состояний, размеры представления которых растут во времени без ограничения~~. Это утверждение нелегко доказать, но в упр. 15.5 показано, что в простых примерах так и происходит.

### Простой одномерный пример

Как было указано выше, оператор Forward для фильтра Калмана отображает одно гауссово распределение на другое, новое гауссово распределение. Применение

этого оператора сводится к вычислению новых значений среднего и матрицы ковариации из предыдущих значений среднего и матрицы ковариации. Для вывода правила обновления в общем (многомерном) случае требуется большой объем выкладок в линейной алгебре, поэтому пока остановимся на очень простом одномерном случае, а позже будут даны результаты для общего случая. Но даже в одномерном случае вычисления являются довольно трудоемкими; тем не менее авторы считают, что с ними следует ознакомиться, поскольку применимость фильтра Калмана слишком тесно связана с математическими свойствами гауссовых распределений.

Во временной модели, которая будет здесь рассматриваться, представлено **случайное блуждание** единственной непрерывной переменной состояния  $X_t$ , зарегистрированное с помощью зашумленных результатов наблюдения  $Z_t$ . Одним из соответствующих примеров может служить показатель “доверия потребителя”, который может быть промоделирован как переменная, подвергающаяся каждый месяц случайному изменению с вероятностью, представленной с помощью гауссова распределения, и измеряемая с помощью опроса случайно выбранных потребителей, в котором также вносится гауссов шум формирования выборки. Предполагается, что распределение априорных вероятностей является гауссовым, с дисперсией  $\sigma_0^2$ :

$$-\frac{1}{2} \left( \frac{(x_0 - \mu_0)^2}{\sigma_0^2} \right)$$

$$P(x_0) = \alpha e$$

(Для упрощения в этом разделе мы будем использовать один и тот же символ  $\alpha$  для обозначения всех констант нормализации.) В модели перехода просто добавляется гауссово возмущение постоянной дисперсии  $\sigma_x^2$  к текущему состоянию:

$$-\frac{1}{2} \left( \frac{(x_{t+1} - x_t)^2}{\sigma_x^2} \right)$$

$$P(x_{t+1} | x_t) = \alpha e$$

Это означает, что в модели восприятия должно быть принято предположение о наличии гауссова шума с дисперсией  $\sigma_z^2$ :

$$-\frac{1}{2} \left( \frac{(z_t - x_t)^2}{\sigma_z^2} \right)$$

$$P(z_t | x_t) = \alpha e$$

Теперь, после получения распределения априорных вероятностей  $P(X_0)$ , мы можем вычислить распределение, прогнозируемое на один этап, с помощью уравнения 15.15:

$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1 | x_0) P(x_0) dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left( \frac{(x_1 - x_0)^2}{\sigma_x^2} \right)} e^{-\frac{1}{2} \left( \frac{(x_0 - \mu_0)^2}{\sigma_0^2} \right)} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left( \frac{\sigma_0^2 (x_1 - x_0)^2 + \sigma_x^2 (x_0 - \mu_0)^2}{\sigma_0^2 \sigma_x^2} \right)} dx_0 \end{aligned}$$

На первый взгляд этот интеграл кажется довольно сложным. Ключом к его упрощению может стать такое замечание, что экспонента представляет собой сумму двух выражений, которые квадратично зависят от  $x_0$ , и поэтому сама экспонента квадра-

тично зависит от  $x_0$ . Но известно, что любое квадратное уравнение  $ax_0^2 + bx_0 + c$  может быть перезаписано как сумма терма, введенного в квадрат,  $a(x_0 - \frac{-b}{2a})^2$ , и остаточного терма  $c - \frac{b^2}{4a}$ , независимого от  $x_0$ , с помощью преобразования, называемого **дополнением квадрата**. Поэтому остаточный терм может быть вынесен за пределы интеграла, что приводит к получению следующего уравнения:

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(c - \frac{b^2}{4a}\right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2}a\left(x_0 - \frac{-b}{2a}\right)^2} dx_0$$

Теперь рассматриваемый интеграл представляет собой обычный интеграл гауссова распределения по всей области его определения, который равен 1. Таким образом, от квадратного уравнения сохраняется лишь его остаточный терм.

Вторым важным этапом является преобразование, выполняемое на основе того наблюдения, что остаточный терм должен иметь квадратичную зависимость от  $x_1$ ; в действительности после его упрощения получаем следующее:

$$-\frac{1}{2} \left( \frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2} \right)$$

$$P(x_1) = \alpha e^{-\frac{1}{2} \left( \frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2} \right)}$$

Таким образом, распределение, прогнозируемое на один этап, представляет собой гауссово распределение с тем же средним  $\mu_0$  и дисперсией, равной сумме первоначальной дисперсии  $\sigma_0^2$  и дисперсии перехода  $\sigma_x^2$ . Даже краткие размышления позволяют понять, что такое соотношение интуитивно вполне оправдано.

Для завершения этапа обновления необходимо обусловить вероятность результатами наблюдения на первом временном этапе, а именно  $z_1$ . Согласно уравнению 15.16, такая операция осуществляется с помощью следующего уравнения:

$$\begin{aligned} P(x_1 | z_1) &= \alpha P(z_1 | x_1) P(x_1) \\ &= \alpha e^{-\frac{1}{2} \left( \frac{(z_1 - x_1)^2}{\sigma_z^2} \right)} \cdot \alpha e^{-\frac{1}{2} \left( \frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2} \right)} \\ &= \alpha e^{-\frac{1}{2} \left( \frac{(z_1 - x_1)^2}{\sigma_z^2} + \frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2} \right)} \end{aligned}$$

Снова объединим экспоненты и дополним квадрат (упр. 15.6), получая следующее:

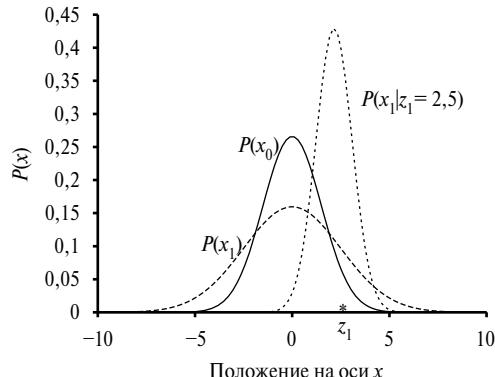
$$P(x_1 | z_1) = \alpha e^{-\frac{1}{2} \left( \frac{(x_1 - \frac{(\sigma_0^2 + \sigma_x^2) z_1 + \sigma_z^2 \mu_0}{\sigma_0^2 + \sigma_x^2 + \sigma_z^2})^2}{(\sigma_0^2 + \sigma_x^2) \sigma_z^2 / (\sigma_0^2 + \sigma_x^2 + \sigma_z^2)} \right)} \quad (15.17)$$

Таким образом, после одного цикла обновления будет получено новое гауссово распределение для переменной состояния.

На основании гауссовой формулы, приведенной в уравнении 15.17, можно определить, что новые значения среднего и среднеквадратичного отклонения можно вычислить на основе старых значений среднего и среднеквадратичного отклонения следующим образом:

$$\begin{aligned}\mu_{t+1} &= \frac{(\sigma_t^2 + \sigma_x^2) z_{t+1} + \sigma_z^2 \mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \\ \sigma_{t+1}^2 &= \frac{(\sigma_t^2 + \sigma_x^2) \sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}\end{aligned}\quad (15.18)$$

На рис. 15.6 показан один цикл обновления для конкретных значений модели перехода и модели восприятия.



*Рис. 15.6. Этапы цикла обновления фильтра Калмана для случайного блуждания с априорной вероятностью, заданной параметрами  $\mu_0 = 0.0$  и  $\sigma_0 = 1.0$ , шумом перехода, заданным  $\sigma_x = 2.0$ , шумом восприятия, заданным  $\sigma_z = 1.0$ , и первым результатом наблюдения  $z_1 = 2.5$  (это значение показано звездочкой на оси  $x$ ). Теперь следует отметить, что под влиянием шума перехода предсказание  $P(x_1)$  слгаживается относительно  $P(x_0)$ . Заслуживает также внимания то, что среднее апостериорной вероятности  $P(x_1|z_1)$  находится немного левее от результата наблюдения  $z_1$ , поскольку это среднее представляет собой взвешенное среднее от предсказания и наблюдения*

Приведенная выше пара уравнений играет точно такую же роль, как и общее уравнение фильтрации, 15.3, или уравнение фильтрации НММ, 15.10. Но в связи с особым характером гауссовых распределений эти уравнения обладают также некоторыми интересными дополнительными свойствами. Во-первых, можно интерпретировать вычисление нового значения среднего  $\mu_{t+1}$  как вычисление взвешенного среднего от новых результатов наблюдения  $z_{t+1}$  и старого значения среднего  $\mu_t$ . Если результаты наблюдения являются ненадежными, то значение  $\sigma_z^2$  увеличивается и мы придаём больший вес старому значению среднего, а если ненадежно старое значение среднего (велико значение  $\sigma_z^2$ ) или процесс является в высшей степени непредсказуемым (велико значение  $\sigma_x^2$ ), то придаём больший вес результатам наблюдения. Во-вторых, следует отметить, что обновление для дисперсии  $\sigma_{t+1}^2$  является независимым от результатов наблюдения. Поэтому можно заранее определить путём вычисления, какой должна быть последовательность значений дисперсии. В-третьих, последовательность значений дисперсии быстро сходится к постоянному значению, которое зависит только от  $\sigma_x^2$  и  $\sigma_z^2$ , что способствует существенному упрощению дальнейших вычислений (см. упр. 15.7).

## Общий случай

Приведенные выше результаты анализа иллюстрируют ключевое свойство гауссовых распределений, которое обеспечивает функционирование методов калмановской фильтрации: тот факт, что экспонента находится в квадратичной форме. Указанное свойство относится не только к одномерному случаю; полное многомерное гауссово распределение имеет следующую форму:

$$-\frac{1}{2} \left( (\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}) \right)$$

$$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha e$$

Кроме того, произведение термов в экспоненте ясно показывает, что экспонента также является квадратичной функцией от случайных переменных  $x_i$ , которые относятся к  $\mathbf{x}$ . Как и в одномерном случае, операция обновления фильтрации сохраняет гауссов характер распределения вероятностей состояний.

Вначале определим общую временную модель, применяемую в процедуре калмановской фильтрации. И модель перехода, и модель восприятия позволяют применять линейное преобразование с дополнительным гауссовым шумом. Таким образом, получаем следующее:

$$\begin{aligned} P(\mathbf{x}_{t+1} | \mathbf{x}_t) &= N(\mathbf{F}\mathbf{x}_t, \boldsymbol{\Sigma}_x)(\mathbf{x}_{t+1}) \\ P(\mathbf{z}_t | \mathbf{x}_t) &= N(\mathbf{H}\mathbf{x}_t, \boldsymbol{\Sigma}_z)(\mathbf{z}_t) \end{aligned} \quad (15.19)$$

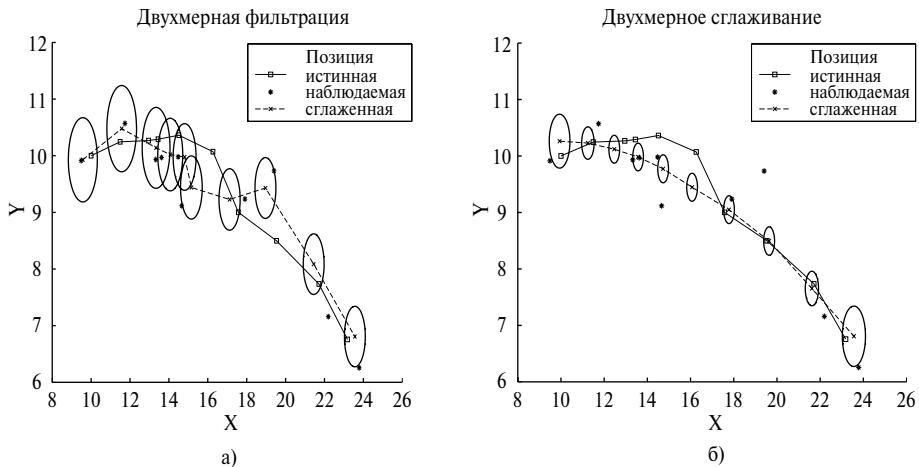
где  $\mathbf{F}$  и  $\boldsymbol{\Sigma}_x$  — матрицы, описывающие линейную модель перехода и ковариацию шума перехода;  $\mathbf{H}$  и  $\boldsymbol{\Sigma}_z$  — соответствующие матрицы для модели восприятия. Теперь уравнения обновления для среднего и ковариации в их полном, ужасающе сложном виде становятся таковыми:

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \boldsymbol{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\boldsymbol{\Sigma}_x\mathbf{F}^\top + \boldsymbol{\Sigma}_x) \end{aligned} \quad (15.20)$$

где  $\mathbf{K}_{t+1} = (\mathbf{F}\boldsymbol{\Sigma}_x\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\boldsymbol{\Sigma}_x\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\mathbf{H}^\top + \boldsymbol{\Sigma}_z)^{-1}$  называется **калмановской матрицей усиления**. Хотите — верьте, хотите — нет, но эти уравнения имеют определенный интуитивный смысл. Например, рассмотрим обновление для оценки значения среднего  $\boldsymbol{\mu}$  для некоторого состояния. Терм  $\mathbf{F}\boldsymbol{\mu}_t$  представляет собой прогнозируемое состояние в момент времени  $t+1$ , поэтому  $\mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$  является прогнозируемым результатом наблюдения. Таким образом, терм  $\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$  соответствует ошибке в прогнозируемых результатах наблюдений. Это значение умножается на  $\mathbf{K}_{t+1}$  для корректировки прогнозируемого состояния, поэтому  $\mathbf{K}_{t+1}$  представляет собой меру того, насколько важными следует считать новые результаты наблюдения применительно к предсказанию. Как и при использовании уравнения 15.18, соблюдается такое свойство, что обновление дисперсии не зависит от результатов наблюдений. Поэтому последовательность значений  $\boldsymbol{\Sigma}_t$  и  $\mathbf{K}_t$  может быть вычислена в автономном режиме, т.е. фактический объем вычислений, требуемых во время оперативного слежения, становится весьма скромным.

Для иллюстрации этих уравнений в действии мы применили их к задаче слежения за объектом, движущимся на плоскости  $X-Y$ . Переменными состоянияя являются  $\mathbf{x} = (X, Y, \dot{X}, \dot{Y})^\top$ , поэтому  $\mathbf{F}$ ,  $\boldsymbol{\Sigma}_x$ ,  $\mathbf{H}$  и  $\boldsymbol{\Sigma}_z$  представляют собой матрицы с размерами  $4 \times 4$ . На рис. 15.7, а показаны истинная траектория, ряд зашумленных результатов

наблюдения и траектория, оцениваемая с помощью калмановской фильтрации, наряду с ковариациями, указанными с помощью контуров единичного среднеквадратичного отклонения. Процесс фильтрации позволяет весьма успешно следить за фактическим перемещением, к тому же, как и предполагалось, дисперсия быстро достигает фиксированной точки.



*Рис. 15.7. Примеры применения уравнений фильтрации и сглаживания: результаты калмановской фильтрации для объекта, движущегося по плоскости X-Y, которые показывают истинную траекторию (слева направо), ряд зашумленных наблюдений и траекторию, оцениваемую с помощью калмановской фильтрации; дисперсия в оценке позиции показана с помощью овалов (а); результаты калмановского сглаживания для той же последовательности результатов наблюдения (б)*

Мы можем вывести не только уравнения фильтрации с помощью линейных гауссовых моделей, но и уравнения сглаживания. Результаты сглаживания показаны на рис. 15.7, б. Обратите внимание на то, как резко сокращается дисперсия в оценке позиции, за исключением концов траектории (объясните, почему), и насколько более гладкой становится оцениваемая траектория.

### Области применения калмановской фильтрации

Методы, основанные на использовании фильтров Калмана и их модификаций, применяются в самых различных приложениях. Одним из “классических” приложений является слежение за самолетами и ракетами с помощью радаров. К такому же типу относятся приложения, в которых осуществляется слежение за подводными лодками и наземными транспортными средствами по звуку, а также визуальное слежение за транспортными средствами и людьми. К немного более узким областям применения относится использование фильтров Калмана для реконструкции траектории частиц по фотографиям, сделанным в пузырьковой камере, и океанских течений по данным измерений, выполненных на поверхности океана со спутников. Но спектр таких приложений далеко выходит за пределы простого отслеживания движений — к ним относится любая система, характеризующаяся непрерывными переменными состояния и зашумленными результатами измерений. К числу подоб-

ных систем относятся целлюлозные фабрики, химические установки, ядерные реакторы, экосистемы растений и национальные экономики.

Тот факт, что калмановская фильтрация может применяться к некоторой системе, не означает, что результаты этого будут действительными или полезными, поскольку используемые при этом допущения (о том, что модели перехода и модели восприятия относятся к типу линейных гауссовых) являются очень строгими. В **расширенном фильтре Калмана** (Extended Kalman Filter — EKF) предпринимается попытка преодолеть нелинейности моделируемой системы. Система является нелинейной, если ее модель перехода нельзя описать с помощью матричного умножения векторов состояния, как в уравнении 15.19. Фильтр EKF действует посредством моделирования системы как локально линейной в области  $\mathbf{x}_t = \boldsymbol{\mu}_t$ , т.е. в той области, где переменные  $\mathbf{x}_t$  равны среднему текущего распределения вероятностей состояний. Этот фильтр хорошо действует применительно к гладким системам с устойчивым поведением и позволяет программе слежения сопровождать и обновлять гауссово распределение вероятностей состояния, которое является приемлемой аппроксимацией истинной апостериорной вероятности.

А что подразумевается под системой, которая “не является гладкой” или “поведение которой неустойчиво”? Формально под этим подразумевается, что отклик системы в области, “близкой” (согласно ковариации  $\boldsymbol{\Sigma}_t$ ) к текущему среднему  $\boldsymbol{\mu}_t$ , показывает существенную нелинейность. Чтобы понять суть этого описания неформально, рассмотрим пример слежения за птицей, которая летит через джунгли. Иногда создается впечатление, что птица направляется на высокой скорости прямо на ствол дерева. Фильтр Калмана (обычный или расширенный) позволяет получить только гауссово предсказание местонахождения птицы, притом что среднее соответствующего гауссова распределения будет находиться напротив центра ствола, как показано на рис. 15.8, а. Но более приемлемая модель полета птицы, с другой стороны, должна предсказывать ее действия по уклонению от удара об ствол путем поворота в одну сторону или в другую, как показано на рис. 15.8, б. Такая модель является в высшей степени нелинейной, поскольку птица принимает решение по уклонению от удара внезапно, в зависимости от того, где именно она находится по отношению к стволу.

Очевидно, что для работы с примерами, подобными этому, требуется более выразительный язык представления поведения моделируемой системы. В сообществе специалистов по теории управления, для которых в таких задачах, как маневры самолета по предотвращению столкновения, возникают аналогичные сложности, разработан стандартный подход — **переключательные фильтры Калмана**. В этом подходе предусмотрена одновременная эксплуатация нескольких фильтров Калмана, в каждом из которых используются разные модели систем, например, в одном из них моделируется прямой полет, в другом — резкий поворот налево, а в третьем — резкий поворот направо. При этом используется взвешенная сумма предсказаний, где вес зависит от того, насколько полно данные каждого фильтра совпадают с текущими данными. Как показано в следующем разделе, такой подход представляет собой частный случай общей модели динамической байесовской сети, созданной путем введения дискретной переменной состояния “маневра” в сеть, показанную на рис. 15.5. Переключательные фильтры Калмана рассматриваются дополнительно в упр. 15.5.

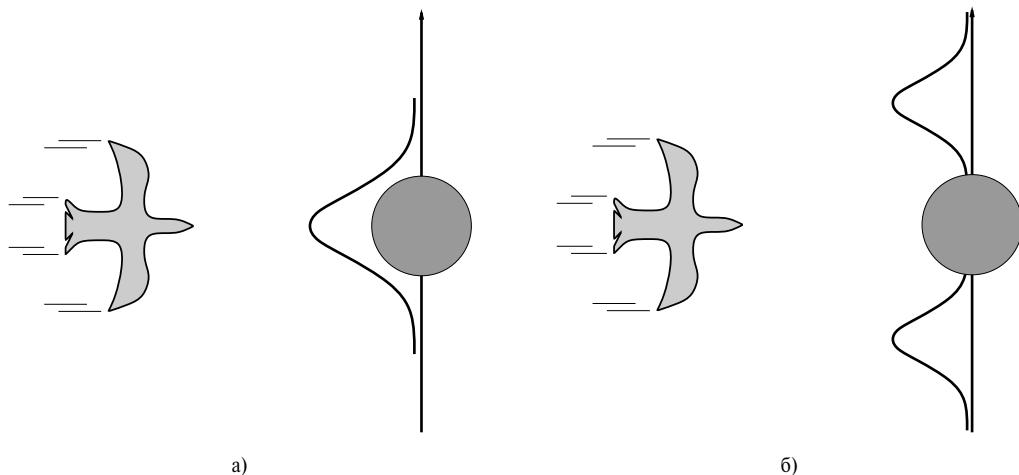


Рис. 15.8. Пример с птицей, летящей прямо на дерево (вид сверху): фильтр Калмана предсказывает местонахождение птицы с использованием единственного гауссова распределения, центр которого находится напротив препятствия (а); более реалистичная модель допускает выполнение птицей действия по уклонению от препятствия, предсказывая, что птица облетит препятствие с одной стороны или с другой (б)

## 15.5. ДИНАМИЧЕСКИЕ БАЙЕСОВСКИЕ СЕТИ

Динамической байесовской сетью, или сокращенно DBN (Dynamic Bayesian Network), называется байесовская сеть, которая представляет временную вероятностную модель такого типа, как описано в разделе 15.1. В данной главе уже рассматривались примеры сетей DBN: сеть для задачи с зонтиком (см. рис. 15.2) и сеть фильтра Калмана (см. рис. 15.5). Вообще говоря, каждый временной срез сети DBN может иметь любое количество переменных состояния  $\mathbf{x}_t$  и переменных свидетельства  $\mathbf{e}_t$ . Для упрощения мы будем предполагать, что переменные и связи между ними точно тиражируются от среза к срезу и что сеть DBN представляет марковский процесс первого порядка, так что каждая переменная может иметь родительские переменные только в своем собственном временном срезе или в непосредственно предшествующем временном срезе.

Должно быть очевидно, что любая скрытая марковская модель может быть представлена в виде сети DBN с единственной переменной состояния и с единственной переменной свидетельства. Справедливо также утверждение, что каждая сеть DBN с дискретными переменными может быть представлена в виде модели HMM; как было описано в разделе 15.3, можно скомбинировать все переменные состояния в сети DBN в одну переменную состояния, значениями которой являются все возможные кортежи значений отдельных переменных состояния. Итак, если каждая модель HMM представляет собой сеть DBN, а каждая сеть DBN может быть преобразована в модель HMM, то в чем состоит различие между ними? Это различие заключается в том, что благодаря декомпозиции состояния сложной системы на составляющие его переменные сеть DBN позволяет воспользоваться преимуществами разреженности временной вероятностной модели. Предположим, например, что в се-

ти DBN имеется 20 булевых переменных состояния, каждая из которых имеет три родительских переменных в предшествующем срезе. В таком случае модель перехода DBN включает  $20 \times 2^3 = 160$  вероятностей, а соответствующая модель HMM имеет  $2^{20}$  состояний и поэтому  $2^{40}$  (или примерно триллион) вероятностей в матрице перехода. Такая ситуация неприемлема по меньшей мере по трем причинам: во-первых, требуется гораздо больше пространства для самой модели HMM; во-вторых, из-за огромной матрицы перехода вероятностный вывод с помощью модели HMM становится гораздо более дорогостоящим; и, в-третьих, из-за проблем, связанных с изучением такого огромного количества параметров, чистая модель HMM становится не-пригодной для решения крупных задач. Связь между сетями DBN и моделями HMM примерно аналогична связи между обычными байесовскими сетями и полностью табулированными совместными распределениями.

Как уже было описано выше, каждая модель с фильтром Калмана может быть представлена в виде сети DBN с непрерывными переменными и линейными гауссовыми распределениями условных вероятностей (см. рис. 15.5). Согласно сведениям, приведенным в конце предыдущего раздела, должно быть очевидно, что не каждая сеть DBN может быть представлена с помощью модели с фильтром Калмана. В фильтре Калмана текущее распределение вероятностей состояния всегда представляет собой единственное многомерное гауссово распределение, т.е. распределение с единственным “максимумом”, расположенным в определенном месте. Сети DBN, с другой стороны, позволяют моделировать произвольные распределения. Такая гибкость является весьма существенным подспорьем для многих реальных приложений. Рассмотрим, например, текущее местонахождение ключей от дома, принадлежащих некоторому лицу. Они могут находиться в его кармане, на ночном столике, в ящике кухни или торчать в замочной скважине входной двери. Единственный гауссов максимум, который охватывает распределения вероятностей нахождения ключей во всех этих местах, назначил бы значительную вероятность тому предположению, что ключи находятся где-то в промежуточной позиции, например висят прямо в воздухе в прихожей. Таким образом, реальный мир, характеризующийся наличием целенаправленных агентов, препятствий и тупиков, приводит к созданию “нелинейности” и поэтому требует применения сочетаний дискретных и непрерывных переменных для того, чтобы можно было создавать приемлемые модели.

## Процедура создания сетей DBN

Для того чтобы создать сеть DBN, необходимо определить три вида информации: распределение априорных вероятностей по переменным состояния,  $P(\mathbf{x}_0)$ ; модель перехода,  $P(\mathbf{x}_{t+1} | \mathbf{x}_t)$ , и модель восприятия,  $P(\mathbf{e}_t | \mathbf{x}_t)$ . Чтобы задать модель перехода и модель восприятия, необходимо определить топологию связей между последовательными срезами, а также между переменными состояния и свидетельства. Поскольку предполагается, что модели перехода и восприятия являются стационарными (одинаковыми для всех  $t$ ), удобнее всего задать их для первого среза. Например, полная спецификация DBN для мира задачи с зонтиком дана с помощью сети с тремя узлами, показанной на рис. 15.9, а. На основании этой спецификации по мере необходимости может быть создана полная (полубесконечная) сеть DBN путем копирования первого среза.

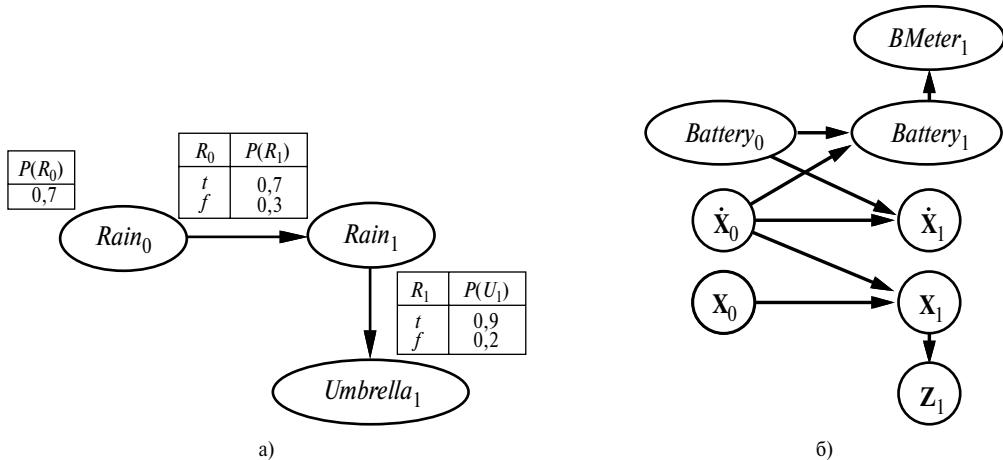


Рис. 15.9. Примеры создания сетей DBN: спецификация распределения априорных вероятностей модели перехода и модели восприятия для сети DBN задачи с зонтиком; предполагается, что все последующие срезы являются копиями среза 1 (а); простая сеть DBN для моделирования движения робота на плоскости X-Y (б)

Теперь рассмотрим более интересный пример: текущий контроль за роботом с аккумуляторным питанием, который движется на плоскости X-Y (о нем впервые шла речь в разделе 15.1). Прежде всего необходимо определить переменные состояния, в состав которых мы включим и переменную  $\mathbf{x}_t = (X_t, Y_t)$ , обозначающую положение, и переменную  $\dot{\mathbf{x}}_t = (\dot{X}_t, \dot{Y}_t)$ , обозначающую скорость. Предполагается, что для измерения координат положения используется определенный метод (возможно, фиксированная телекамера или бортовая система GPS (Global Positioning System — глобальная система позиционирования)), позволяющий получить результаты измерений  $\mathbf{z}_t$ . Положение робота в следующем временном интервале зависит от текущего положения и скорости, как и при использовании стандартной модели с фильтром Калмана. Скорость в следующем временном интервале зависит от текущей скорости и состояния зарядки аккумулятора. Введем переменную  $Battery_t$  для обозначения фактического уровня зарядки аккумулятора, родительскими переменными которой являются предыдущий уровень зарядки аккумулятора и скорость, а также введем переменную  $BMeter_t$ , которая измеряет уровень зарядки аккумулятора. В результате будет получена исходная модель, показанная на рис. 15.9, б.

Характер модели восприятия для переменной  $BMeter_t$  заслуживает более глубокого анализа. Для упрощения предположим, что переменные  $Battery_t$  и  $BMeter_t$  могут принимать дискретные значения от 0 до 5, во многом аналогично измерительному прибору в типичном портативном компьютере, который применяется для контроля за аккумулятором. Если этот прибор всегда дает точные показания, то таблица условных вероятностей  $P(BMeter_t | Battery_t)$  должна содержать вероятности 1.0 в элементах, расположенных “вдоль диагонали”, и вероятности 0.0 во всех других элементах. Но в действительности в результате измерения всегда проникает шум, поэтому при использовании непрерывных измерений вместо этих результатов

может использоваться гауссово распределение с небольшой дисперсией<sup>4</sup>. Применительно к дискретным переменным, рассматриваемым в данном примере, гауссово распределение можно аппроксимировать с помощью распределения, в котором снижение вероятности ошибки соответствует реальной ситуации, поэтому вероятность крупной ошибки весьма мала. Мы будем использовать термин **гауссова модель ошибки** применительно и к непрерывной, и к дискретной версиям.

Те, кто имеет практический опыт работы в области робототехники, компьютеризированного управления процессами или в области применения других форм автоматического сбора информации, охотно подтверждают тот факт, что небольшие количества измерительного шума часто являются не самыми серьезными проблемами. Гораздо важнее то, что настоящие датчики отказывают, а когда датчик отказывает, он не всегда посылает сигнал с сообщением: “Между прочим, данные, которые я собираюсь вам отправить, можно считать бессмысленными”. Вместо этого он просто передает бессмыслицу. Отказом простейшего типа является **временный отказ**, при котором датчик время от времени передает бессмысленные данные. Например, может оказаться, что датчик уровня зарядки аккумулятора имеет свойство передавать нулевое значение каждый раз, когда робот ударяется о препятствие, даже если аккумулятор полностью заряжен.

Рассмотрим, что произойдет при возникновении временного отказа, если используется гауссова модель ошибок, которая не приспособлена к таким отказам. Предположим, например, что робот спокойно ожидает и наблюдает за 20 последовательными показаниями датчика аккумулятора, равными 5. Затем датчик аккумулятора допускает временный сбой и передает следующее показание:  $B_{Meter_{21}}=0$ . К какому решению должна привести нас простая гауссова модель ошибки применительно к этому значению  $B_{Battery_{21}}$ ? Согласно правилу Байеса, ответ на этот вопрос зависит и от модели восприятия  $P(B_{Meter_{21}}=0 | B_{Battery_{21}})$ , и от предсказания  $P(B_{Battery_{21}} | B_{Meter_{1:20}})$ . Если вероятность большой ошибки датчика является значительно менее правдоподобной, чем вероятность перехода в состояние  $B_{Battery_{21}}=0$ , даже если последняя ситуация весьма неправдоподобна, то в распределении апостериорных вероятностей будет присвоена высокая вероятность той ситуации, что аккумулятор разряжен. Если же в момент времени  $t=22$  будет получено еще одно показание, равное нулю, то такой вывод станет почти полностью безоговорочным. А после того, как этот временный отказ исчезнет и показания вернутся к 5, начиная с момента  $t=23$  и продолжаясь в последующие моменты, то оценка уровня зарядки аккумулятора, как по волшебству, быстро вернется к 5. Такой ход событий проиллюстрирован на верхней кривой, приведенной на рис. 15.10, а, где показано ожидаемое изменение значения переменной  $B_{Battery_t}$  во времени при использовании дискретной гауссовой модели ошибки.

Несмотря на последующее восстановление правильных показаний, есть такой момент времени ( $t=22$ ), в котором робот получил сообщение о разрядке аккумулятора, а в такой ситуации он должен, в принципе, передать сигнал тревоги и отключиться. Итак, к сожалению, чрезмерно упрощенная модель восприятия завела робота в тупик. Как можно исправить такое положение? Рассмотрим знакомый многим

<sup>4</sup> Строго говоря, гауссово распределение не совсем подходит, поскольку в нем присваивается ненулевая вероятность большим отрицательным уровням зарядки аккумулятора. Иногда для переменных, область определения которых ограничена, лучше подходит **бета-распределение**.

пример, в котором при вождении автомобиля на резких поворотах или крутых подъемах иногда загорается сигнал “топливный бак пуст”. Но, вместо того чтобы искать телефон аварийной службы, водитель вспоминает, что датчик уровня топлива иногда допускает очень большую ошибку, когда топливо плецется в баке. Мораль этой истории состоит в следующем:  для того чтобы система правильно учитывала отказы датчика, модель восприятия должна допускать вероятность его отказа.

В модели отказа простейшего вида для датчика допускается определенная вероятность того, что датчик может выдать полностью неправильное значение, независимо от истинного состояния мира. Например, если прибор на аккумуляторе на время отказывает, возвращая значение 0, то можно ввести значение

$$P(BMeter_t=0 | Battery_t=5) = 0.03$$

которое, очевидно, значительно больше, чем вероятность, присваиваемая при использовании простой гауссовой модели ошибки. Назовем соответствующую модель  **моделью временного отказа**. Как она может помочь, если мы столкнемся с показанием датчика, равным 0? При условии, что прогнозируемая вероятность полной разрядки аккумулятора, согласно полученным до сих пор показаниям, гораздо меньше 0.03, то наилучшим объяснением причин наблюдения  $BMeter_{21}=0$  является то, что произошел временный отказ датчика. Интуитивно ясно, что можно рассматривать уверенность в истинности данных об уровне зарядки аккумулятора как имеющую определенную долю “инерции”, которая позволяет преодолеть временные сбои в показаниях датчика. Верхняя кривая на рис. 15.10, б показывает, что модель временного отказа позволяет преодолевать временные отказы без катастрофического изменения в представлениях об истинности данных.

На этом описание временных отказов оканчивается. А что будет, если отказ датчика окажется постоянным? К сожалению, отказы такого типа встречаются слишком часто. Если датчик возвратит 20 показаний со значением 5, за которыми последует 20 показаний со значением 0, то применение модели временного отказа датчика, описанной в предыдущем абзаце, приведет к тому, что робот постепенно придет к выводу, что его аккумулятор разряжен, тогда как в действительности мог произойти отказ датчика. Нижняя кривая, приведенная на рис. 15.10, б, показывает “траекторию” изменения уверенности в истинности показаний датчика для этого случая. Ко времени  $t=25$  (после получения пяти показаний датчика, равных 0) робот приходит к выводу, что его аккумулятор разряжен. Безусловно, мы предпочли бы, чтобы робот приобрел уверенность в том, что неисправен прибор на его аккумуляторе (если такая ситуация действительно является гораздо более вероятной).

Нет ничего удивительного в том, что для учета постоянных отказов требуется  **модель постоянного отказа**, которая описывает, как датчик ведет себя при нормальных условиях и после отказа. Для этого необходимо дополнить скрытое состояние системы дополнительной переменной, скажем,  $BMBroken$ , которая описывает состояние прибора аккумулятора. Постоянство отказа может быть промоделировано дугой, связывающей переменные  $BMBroken_0$  и  $BMBroken_1$ . Такая  **дуга постоянства** имеет таблицу условных вероятностей, которая задает на каждом временном интервале малую вероятность отказа, допустим 0.001, но определяет, что датчик после выхода из строя остается неисправным. Если датчик исправен, то модель восприятия для переменной  $BMeter$  идентична модели временного отказа, а после того

как датчик становится неисправным, эта модель указывает, что значение  $B_{Meter}$  всегда равно 0, независимо от фактической зарядки аккумулятора.

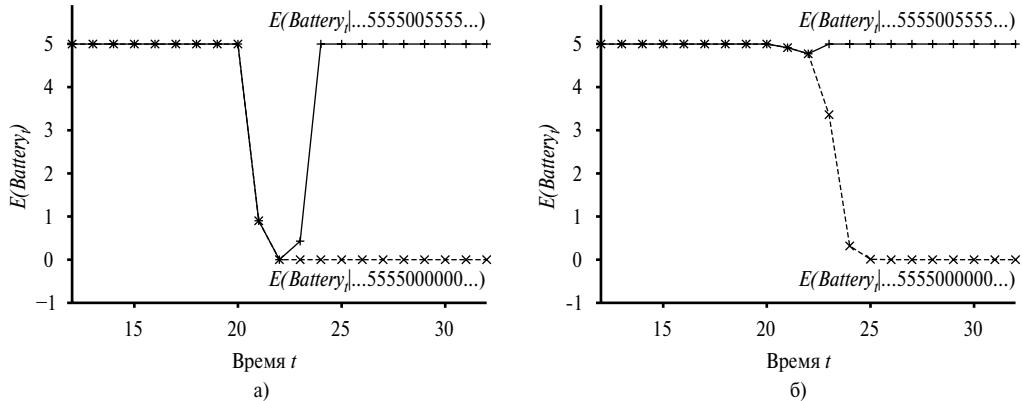


Рис. 15.10. Пример применения модели временного отказа: верхняя кривая — траектория ожидаемого значения переменной  $Battery_t$  для последовательности наблюдений, состоящей из всех значений 5, кроме показаний, равных 0, в моменты времени  $t=21$  и  $t=22$ , если используется простая гауссова модель ошибки; нижняя кривая — траектория, при которой результаты наблюдения остаются на уровне 0, начиная с момента времени  $t=21$  (а); такой же эксперимент, проведенный с использованием модели временного отказа (б); обратите внимание на то, что временный отказ преодолевается успешно, а постоянный приводит к излишне пессимистическому поведению

Модель постоянного отказа для датчика аккумулятора показана на рис. 15.11, а. Показатели ее производительности при двух последовательностях данных (временный сбой и постоянный отказ) приведены на рис. 15.11, б. В отношении этих кривых необходимо сделать несколько замечаний. Прежде всего, в случае временного сбоя вероятность того, что датчик вышел из строя, существенно повышается после второго показания со значением 0, но немедленно падает вновь до нуля после получения результатов наблюдения 5. Далее, в случае постоянного отказа вероятность того, что датчик неисправен, быстро повышается почти до 1 и остается на этом уровне. Наконец, как только становится известно, что датчик стал неисправным, робот может лишь руководствоваться предположением, что его аккумулятор разряжается с “обычной” скоростью, как показывает постепенно снижающийся уровень  $E(Battery_t | \dots)$ .

В приведенном выше описании мы лишь слегка коснулись поверхности проблемы представления сложных процессов. Применяемое на практике разнообразие моделей перехода является буквально огромным и охватывает такие разные направления, как моделирование эндокринной системы человека и моделирование потока, состоящего из большого количества автомобилей, движущихся по скоростному шоссе. Создание моделей восприятия также представляет собой обширную самостоятельную область, но практика показала, что динамические байесовские сети позволяют явно представить даже такие тонкие феномены, как дрейф датчика, внезапная раскалибровка и влияние на показания датчика внешних условий (таких как погода).

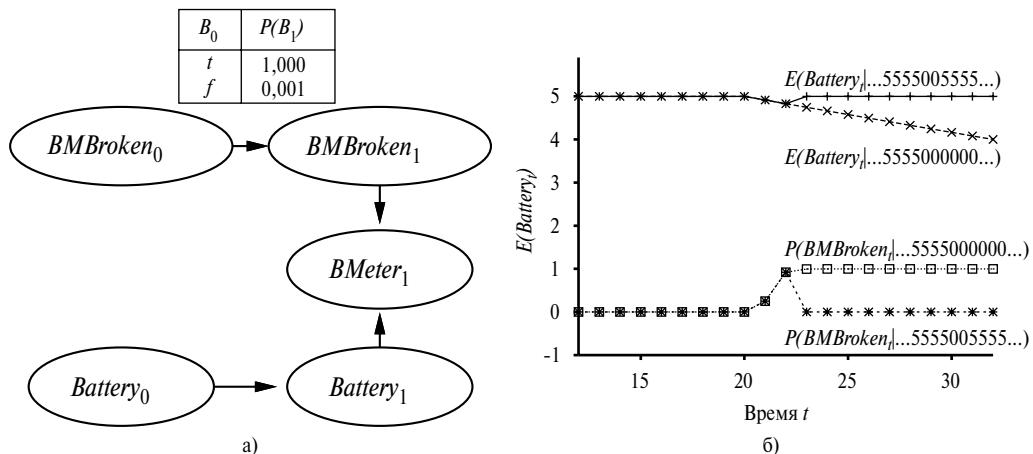


Рис. 15.11. Пример применения модели постоянного отказа: фрагмент сети DBN, в котором показаны переменные состояния датчика, необходимые для моделирования постоянного отказа датчика аккумулятора (а); верхние кривые — траектории ожидаемого значения переменной  $Battery_t$  для последовательностей наблюдений, характерных для “временного отказа” и “постоянного отказа”; нижние кривые — траектории вероятностей для переменной  $BM Broken$  при наличии двух указанных последовательностей наблюдений (б)

### Точный вероятностный вывод в сетях DBN

Кратко рассмотрев некоторые идеи, касающиеся представления сложных процессов в виде сетей DBN, перейдем теперь к вопросу вероятностного вывода. В определенном смысле на этот вопрос уже был получен ответ: динамические байесовские сети прежде всего представляют собой байесовские сети, а мы уже имеем алгоритмы для осуществления вероятностного вывода в байесовских сетях. При наличии последовательности наблюдений может быть создано представление сети DBN в виде полной байесовской сети путем повторения временных срезов до тех пор, пока сеть не станет достаточно большой, чтобы в ней можно было учесть все наблюдения, как показано на рис. 15.12. Такой метод называется **развертыванием**. (С формальной точки зрения сеть DBN эквивалентна полубесконечной сети, полученной путем развертывания в одну сторону до бесконечности. Но временные срезы, вводимые за пределами последнего наблюдения, не оказывают влияния на вероятностные выводы в пределах периода наблюдения и поэтому могут быть исключены.) После того как сеть DBN развернута, в ней может использоваться любой из алгоритмов вероятностного вывода (алгоритм с устранением переменной, методы соединенного дерева и т.д.), описанные в главе 14.

К сожалению, непродуманное применение развертывания не всегда оказывается достаточно эффективным. В частности, если требуется выполнение фильтрации или сглаживания с использованием длинной последовательности наблюдений  $e_{1:t}$ , то для развернутой сети потребуется пространство  $O(t)$ , а рост ее по мере добавления новых результатов наблюдений будет происходить неограниченно. Более того, если после каждого добавления новых результатов наблюдения будет просто вновь вызываться на выполнение алгоритм вероятностного вывода, то затраты времени на вероятностный вывод в расчете на каждое обновление также будут расти пропорционально  $O(t)$ .

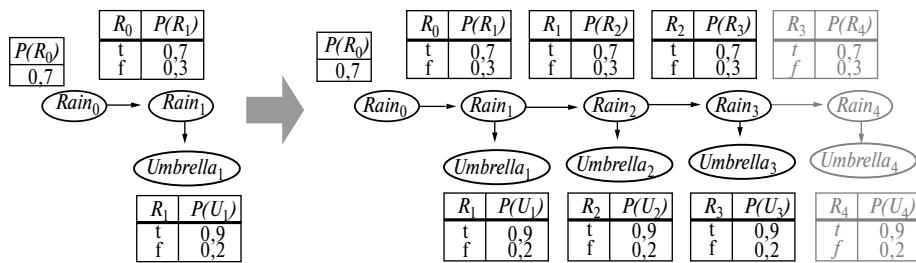


Рис. 15.12. Разворачивание динамической байесовской сети: для размещения результатов последовательности наблюдений дублируются временные срезы (обозначенные овалами с жирными контурами). Следующие срезы не влияют на вероятностные выводы в период наблюдения

Еще раз обратившись к разделу 15.2, можно отметить, что постоянных затрат времени и пространства в расчете на каждое обновление при фильтрации можно достичь, если существует возможность выполнять вычисление в рекурсивной форме. По сути обновление результатов фильтрации в уравнении 15.3 осуществляется по принципу исключения путем суммирования переменных состояния, относящихся к предыдущему временному этапу, что позволяет получить распределение для нового временного этапа. Исключение переменных путем суммирования представляет собой именно то, что выполняет алгоритм **устранения переменной** (см. листинг 14.2), и, как оказалось, применение процедуры устранения переменной к переменным во временном порядке точно моделирует функционирование рекурсивного обновления результатов фильтрации в уравнении 15.3. В модифицированном алгоритме предусмотрено одновременное хранение в памяти не больше двух временных срезов: начиная со среза 0, добавляем срез 1, затем исключаем путем суммирования срез 0, после этого добавляем срез 2, на следующем этапе исключаем путем суммирования срез 1 и т.д. Такая организация вычислений позволяет добиться постоянных затрат пространства и времени в расчете на каждое обновление результатов фильтрации. (Такой же производительности можно достичь путем введения соответствующих модификаций в алгоритм соединенного дерева.) В упр. 15.10 предлагается проверить это утверждение на примере сети для задачи с зонтиком.

До сих пор рассматривались только преимущества рекурсивного подхода, но он имеет и недостатки: как оказалось, “постоянные” значения временной и пространственной сложности каждой операции обновления почти во всех случаях экспоненциально зависят от количества переменных состояния. В связи с этим в ходе осуществления процесса устранения переменной количество факторов возрастает так, что в их состав начинают входить все переменные состояния (или, точнее, все те переменные состояния, которые имеют родительские переменные в предыдущем временном срезе). Максимальный размер фактора составляет  $O(d^{n+1})$ , а стоимость обновления измеряется как  $O(d^{n+2})$ .

Безусловно, такие значения намного меньше по сравнению со стоимостью обновления HMM, которая составляет  $O(d^{2n})$ , но они все еще неприемлемы при наличии большого количества переменных. С этим обескураживающим фактом довольно трудно смириться, поскольку он означает, что ~~даже несмотря на то, что сети DBN могут использоваться для представления очень сложных временных процессов с многочисленными переменными, характеризующимися разрозненными связями между~~

*ними, не существует возможности эффективно и точно формировать вероятностные рассуждения об этих процессах.* Сама модель DBN, которая представляет априорное совместное распределение по всем переменным, может быть разложена на составляющие ее таблицы условных вероятностей, но обусловленное последовательностью наблюдений апостериорное совместное распределение (т.е. прямое сообщение), как правило, не поддается разбиению на факторы. До сих пор еще никому не удалось найти способа решения этой проблемы, несмотря на то, что многие важные области науки и техники получили бы огромную выгоду благодаря такому решению. Поэтому нам приходится обращаться к приближенным методам.

### Приближенный вероятностный вывод в сетях DBN

В главе 14 были описаны два алгоритма аппроксимации — взвешивание с учетом правдоподобия (см. листинг 14.5) и метод Монте-Карло на основе цепи Маркова (алгоритм MCMC, см. листинг 14.6). Из этих двух алгоритмов наиболее легко к контексту DBN адаптируется первый алгоритм. Тем не менее, как будет показано ниже, в стандартный алгоритм взвешивания с учетом правдоподобия необходимо внести несколько усовершенствований, прежде чем появится практически применимый метод.

Напомним, что метод взвешивания с учетом правдоподобия действует по принципу осуществления в топологическом порядке выборок в узлах сети, не являющихся узлами свидетельства, и взвешивания каждой выборки с учетом правдоподобия того, что она соответствует наблюдаемым переменным свидетельства. Как и при использовании точных алгоритмов, алгоритм взвешивания с учетом правдоподобия можно было бы применить непосредственно к развернутой сети DBN, но по мере увеличения длины последовательностей наблюдений это привело бы к возникновению таких же сложностей, связанных с увеличением требований ко времени и пространству в расчете на каждое обновление. Проблема состоит в том, что в стандартном алгоритме каждая выборка обрабатывается последовательно, по всей сети. Вместо этого можно просто пропустить через сеть DBN все  $N$  выборок вместе, проходя каждый раз через один временной срез. Этот модифицированный алгоритм имеет такую же общую форму, как и другие алгоритмы фильтрации, но в нем в качестве прямого сообщения используется множество из  $N$  выборок. Поэтому первым ключевым усовершенствованием должно быть то, чтобы в этом алгоритме *в качестве приближенного представления текущего распределения вероятностей состояния использовались сами выборки*. Такая организация работы соответствует требованию обеспечения “постоянных” затрат времени в расчете на каждое обновление, хотя само это постоянное значение зависит от количества выборок, необходимых для достижения приемлемой аппроксимации истинного апостериорного распределения. Кроме того, нет необходимости развертывать сеть DBN, поскольку в памяти требуется держать только текущий временной срез и следующий временной срез.

В описании метода взвешивания с учетом правдоподобия, приведенного в главе 14, было указано, что точность алгоритма снижается, если переменные свидетельства находятся в “прямом направлении” от переменных, по которым осуществляется выборка, поскольку в таком случае выборки формируются, не испытывая какого-либо влияния со стороны свидетельства. Рассматривая типичную структуру сети DBN (допустим, сети DBN для задачи с зонтиком на рис. 15.12), можно убедиться в том, что в действительности выборку ранее полученных переменных состояния

можно осуществлять, не пользуясь полученным в дальнейшем свидетельством. Проанализировав фактически этот вопрос более внимательно, можно обнаружить, что ни у одной из переменных состояния не имеется среди ее предков ни одной переменной свидетельства! Поэтому, хотя вес каждой выборки зависит от свидетельства, фактически сформированное множество выборок является полностью независимым от свидетельства. Например, даже если директор ходит с зонтиком каждый день, в процессе осуществления выборки все еще может создаваться впечатление, что солнечные дни не кончаются. С точки зрения практики это означает, что доля выборок, остающихся достаточно близкими к фактическому ряду событий, падает экспоненциально со значением  $t$ , т.е. с длиной последовательности наблюдений; иными словами, чтобы поддерживать заданный уровень точности, необходимо увеличивать количество выборок экспоненциально в зависимости от  $t$ . Учитывая то, что алгоритм фильтрации, работающий в реальном времени, может использовать лишь фиксированное количество выборок, на практике происходит то, что после небольшого количества этапов обновления ошибка становится весьма значительной.

Очевидно, что требуется лучшее решение. Поэтому второе важное нововведение состоит в том, что ~~множество выборок в основном следует формировать в областях пространства состояний, характеризующихся высокой вероятностью~~. Такую задачу можно выполнить, отбрасывая выборки, которые, согласно наблюдениям, имеют очень малый вес, вместе с тем увеличивая количество выборок, имеющих большой вес. Благодаря этому появляется возможность создавать множества выборок, которые остаются достаточно близкими к действительным данным. Если выборки рассматриваются как информационные ресурсы для моделирования распределения апостериорных вероятностей, то имеет смысл формировать больше выборок в тех областях пространства состояний, где апостериорная вероятность выше.

Для решения именно этой задачи предназначено семейство алгоритмов, называемых алгоритмами ~~фильтрации частиц~~. Метод фильтрации частиц действует следующим образом: прежде всего создается популяция из  $N$  выборок путем формирования выборок из распределения априорных вероятностей в момент времени 0,  $P(\mathbf{x}_0)$ , затем, как описано ниже, для каждого временного интервала повторяется цикл обновления.

- Каждая выборка распространяется в прямом направлении путем формирования выборки значения переменной следующего состояния  $\mathbf{x}_{t+1}$ . Для этого в качестве выборки берется текущее значение  $\mathbf{x}_t$  и используется модель перехода  $P(\mathbf{x}_{t+1} | \mathbf{x}_t)$ .
- Каждая выборка взвешивается с учетом правдоподобия, которое она присваивает новому свидетельству,  $P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1})$ .
- В этой популяции выборок снова осуществляется формирование выборки для создания новой популяции из  $N$  выборок. Каждая новая выборка берется из текущей популяции; вероятность того, что будет получена конкретная выборка, пропорциональна ее весу. Данные о весах новых выборок уничтожаются.

Этот алгоритм подробно показан в листинге 15.3, а пример его применения к сети DBN для задачи с зонтиком приведен на рис. 15.13.

**Листинг 15.3.** Алгоритм фильтрации частиц, реализованный как рекурсивная операция обновления данных о состоянии (множества выборок). Каждый из этапов формирования выборок состоит в том, что формируется выборка значений соответствующих переменных временного среза в топологическом порядке, во многом так же, как и в процедуре **Prior-Sample**. Операция **Weighted-Sample-With-Replacement** может быть реализована так, чтобы она выполнялась за ожидаемое время  $O(N)$

---

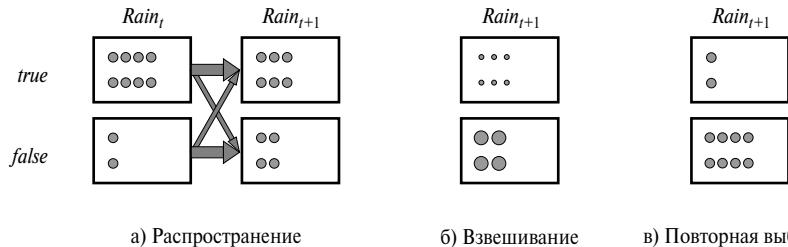
```

function Particle-Filtering(e,  $N$ ,  $dbn$ ) returns множество выборок
    для следующего временного интервала
        inputs: e, новое полученное свидетельство
             $N$ , количество выборок, которые должны сопровождаться
            алгоритмом
             $dbn$ , сеть DBN с распределением априорных вероятностей  $\mathbf{P}(\mathbf{x}_0)$ ,
            моделью перехода  $\mathbf{P}(\mathbf{x}_1|\mathbf{x}_0)$  и моделью восприятия  $\mathbf{P}(\mathbf{e}_1|\mathbf{x}_1)$ 
        static:  $S$ , вектор выборок с размером  $N$ , первоначально формируемый
            из  $\mathbf{P}(\mathbf{x}_0)$ 
        local variables:  $W$ , вектор весов с размером  $N$ 

        for  $i = 1$  to  $N$  do
             $S[i] \leftarrow$  выборка из  $\mathbf{P}(\mathbf{x}_1|\mathbf{x}_0=S[i])$ 
             $W[i] \leftarrow \mathbf{P}(\mathbf{e}| \mathbf{x}_1=S[i])$ 
         $S \leftarrow$  Weighted-Sample-With-Replacement( $N$ ,  $S$ ,  $W$ )
    return  $S$ 

```

---



*Рис. 15.13. Пример применения цикла обновления алгоритма фильтрации частиц к сети DBN для задачи с зонтиком при  $N=10$ , в котором показаны популяции выборок в каждом состоянии: во время  $t$  обнаруживается, что 8 выборок показывают Rain, а 2 выборки показывают  $\neg$ Rain. Каждая из них распространяется в прямом направлении путем формирования выборок в следующем состоянии через модель перехода. Во время  $t+1$  обнаруживается, что 6 выборок показывают Rain, а 4 выборки показывают  $\neg$ Rain (a); в момент времени  $t+1$  наблюдается  $\neg$ Umbrella. Каждая выборка взвешивается с учетом ее правдоподобия применительно к этому наблюдению, как показывают размеры кружков (б); формируется новое множество из 10 выборок путем случайного выбора со взвешиванием из текущего множества, что приводит к получению 2 выборок, которые показывают Rain, и 8 выборок, которые показывают  $\neg$ Rain (в)*

Рассматривая, что происходит во время одного цикла обновления, можно показать, что этот алгоритм является согласованным (позволяет получить правильные значения вероятностей, если  $N$  стремится к бесконечности). Предполагается, что создание популяции выборок начинается с использования правильного представле-

ния прямого сообщения  $\mathbf{f}_{1:t}$  во время  $t$ . Поэтому, записав выражение  $N(\mathbf{x}_t | \mathbf{e}_{1:t})$  для количества выборок, занимающих состояние  $\mathbf{x}_t$  после обработки наблюдений  $\mathbf{e}_{1:t}$ , получаем следующее соотношение для больших значений  $N$ :

$$N(\mathbf{x}_t | \mathbf{e}_{1:t}) / N = P(\mathbf{x}_t | \mathbf{e}_{1:t}) \quad (15.21)$$

Теперь распространим каждую выборку в прямом направлении, осуществляя формирование выборок значений переменных состояния во время  $t+1$  и взяв для каждой выборки значения во время  $t$ . Количество выборок, достигающих состояния  $\mathbf{x}_{t+1}$  из каждого состояния  $\mathbf{x}_t$ , равно вероятности перехода, умноженной на величину популяции  $\mathbf{x}_t$ , поэтому общее количество выборок, достигающих  $\mathbf{x}_{t+1}$ , будет равно следующему:

$$N(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t) N(\mathbf{x}_t | \mathbf{e}_{1:t})$$

После этого выполним взвешивание каждой выборки по ее правдоподобию применительно к свидетельству во время  $t+1$ . Любая отдельная выборка в состоянии  $\mathbf{x}_{t+1}$  получает вес  $P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1})$ . Это означает, что суммарный вес выборок в состоянии  $\mathbf{x}_{t+1}$  после получения свидетельства  $\mathbf{e}_{t+1}$  определяется таким образом:

$$W(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$$

Далее выполняется этап повторного формирования выборки. Поскольку каждая выборка тиражируется с вероятностью, пропорциональной ее весу, количество выборок в состоянии  $\mathbf{x}_{t+1}$  после повторного формирования выборки пропорционально суммарному весу в состоянии  $\mathbf{x}_{t+1}$  перед повторным формированием выборки, как показано ниже.

$$\begin{aligned} N(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) / N &= \alpha W(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t) N(\mathbf{x}_t | \mathbf{e}_{1:t}) \\ &= \alpha N P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) \quad (\text{согласно 15.21}) \\ &= \alpha' P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) \\ &= P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) \quad (\text{согласно 15.3}) \end{aligned}$$

Поэтому популяция выборок после одного цикла обновления правильно представляет прямое сообщение во время  $t+1$ .

Итак, алгоритм фильтрации частиц является согласованным, но характеризуется ли он достаточной эффективностью? Создается впечатление, что на практике ответ на этот вопрос является положительным: по-видимому, фильтрация частиц позволяет поддерживать хорошую аппроксимацию истинных апостериорных вероятностей с использованием постоянного количества выборок. Тем не менее теория еще не дает такой гарантии; в настоящее время в области фильтрации частиц продолжаются интенсивные исследования. Предложено много вариантов и усовершенствования

ний, а спектр приложений этого алгоритма стремительно растет. Поскольку алгоритм фильтрации представляет собой алгоритм формирования выборок, его можно легко применить к гибридным и непрерывным сетям DBN, что дает возможность воспользоваться этим алгоритмом в таких областях, как отслеживание сложных движущихся образов в видеозаписях [719] и предсказание курсов акций на фондовой бирже [343].

## 15.6. РАСПОЗНАВАНИЕ РЕЧИ

В данном разделе рассматривается одно из наиболее важных приложений временных вероятностных моделей — **распознавание речи**. Задача состоит в том, чтобы выявить последовательность слов, произнесенных говорящим, используя акустический сигнал. Речь — это доминирующая форма общения людей, и поэтому надежное распознавание речи с помощью машин было бы чрезвычайно полезным. Еще более важной задачей является **понимание речи** — идентификация смысла фрагментов речи. Но изучение этой темы мы отложим до главы 22.

Речь — это одно из первых проявлений грубого неоткристализованного мира реальных сенсорных данных, с которыми сталкивается человек после своего рождения. Эти данные являются зашумленными, в буквальном смысле слова: в них может присутствовать не только фоновый шум, но и помехи, возникающие в процессе самого преобразования в осознаваемую человеком форму; слова иногда произносятся по-разному, даже одним и тем же говорящим; различные слова могут звучать одинаково и т.д. По этим причинам со временем возникло понимание того, что распознавание речи должно рассматриваться как одна из задач вероятностного вывода.

На наиболее общем уровне эту задачу вероятностного вывода можно определить следующим образом. Предположим, что *Words* — случайная переменная, пробегающая по всем возможным последовательностям слов, которые могут быть произнесены, а также допустим, что *signal* — наблюдаемая последовательность акустических сигналов. В таком случае наиболее вероятной интерпретацией фрагмента речи является то значение переменной *Words*, которое максимизирует вероятность  $P(\text{words} | \text{signal})$ . Как и во многих других случаях, нам может помочь применение правила Байеса:

$$P(\text{words} | \text{signal}) = \alpha P(\text{signal} | \text{words}) P(\text{words})$$

Выражение  $P(\text{signal} | \text{words})$  называется **акустической моделью**. Эта модель описывает звуки слов, например, говорит о том, что слово “ceiling” (потолок) начинается с мягкого звука “с” и звучит так же, как “sealing” (уплотнение). (Слова, звучащие одинаково, часто называют **омофонами**.) Выражение  $P(\text{words})$  принято называть **языковой моделью**. Эта модель задает априорную вероятность каждого фрагмента речи, например указывает, что последовательность слов “high ceiling” (высокий потолок) является гораздо более вероятной, чем “high sealing” (высокое уплотнение).

Языковые модели, используемые в системах распознавания речи, обычно являются очень простыми. Модель **двухсловных сочетаний**, которая будет описана ниже в данном разделе, задает вероятность каждого слова, которое следует за каждым другим словом. Акустическая модель является гораздо более сложной. В ее основе

лежит важное открытие, сделанное в области **фонологии** (науки о звуках устной речи), согласно которому во всех человеческих языках используется ограниченный набор звуков, называемых **фонемами**, количество которых находится в пределах от 40 до 50. Грубо говоря, фонема — это звук, который соответствует одной гласной или согласной букве, но существуют некоторые сложности; например, некоторые сочетания букв, такие как “th” и “ng”, в английском языке соответствуют единственным фонемам, а некоторые буквы произносятся как разные фонемы в различных контекстах (в качестве примера можно указать букву “a” в словах “rat” и “rate”). В табл. 15.1 перечислены фонемы, используемые в английском языке, с примером для каждой из них. Итак, **фонема** — это наименьший фрагмент звукового сигнала, который имеет различимый смысл для людей, говорящих на конкретном языке. Например, в английском языке фонема “t” в слове “stick” является той же самой, что и фонема “t” в слове “tick”, но в тайском языке они различаются как две отдельные фонемы.

**Таблица 15.1. Фонетический алфавит DARPA, или ARPAbet, в котором перечислены все фонемы, используемые в американском диалекте английского языка. Существует также несколько альтернативных систем обозначения фонем, включая международный фонетический алфавит (International Phonetic Alphabet — IPA), который описывает фонемы всех известных языков**

Гласные		Согласные B-N		Согласные P-Z	
Фонема	Пример	Фонема	Пример	Фонема	Пример
[iy]	<u>beat</u>	[b]	<u>bet</u>	[p]	<u>pet</u>
[ih]	<u>bit</u>	[ch]	<u>Chet</u>	[r]	<u>rat</u>
[eh]	<u>bet</u>	[d]	<u>debt</u>	[s]	<u>set</u>
[ae]	<u>bat</u>	[f]	<u>fat</u>	[sh]	<u>shoe</u>
[ah]	<u>but</u>	[g]	<u>get</u>	[t]	<u>ten</u>
[ao]	<u>bought</u>	[hh]	<u>hat</u>	[th]	<u>thick</u>
[ow]	<u>boat</u>	[hv]	<u>high</u>	[dh]	<u>that</u>
[uh]	<u>book</u>	[jh]	<u>jet</u>	[dx]	<u>butter</u>
[ey]	<u>bait</u>	[k]	<u>kick</u>	[v]	<u>yet</u>
[er]	<u>Bert</u>	[l]	<u>let</u>	[w]	<u>wet</u>
[ay]	<u>buy</u>	[el]	<u>bottle</u>	[wh]	<u>which</u>
[oy]	<u>boy</u>	[m]	<u>met</u>	[y]	<u>yet</u>
[axr]	<u>diner</u>	[em]	<u>bottom</u>	[z]	<u>zoo</u>
[aw]	<u>down</u>	[n]	<u>net</u>	[zh]	<u>measure</u>
[ax]	<u>about</u>	[en]	<u>button</u>		
[ix]	<u>roses</u>	[ng]	<u>sing</u>		
[aa]	<u>cot</u>	[eng]	<u>washing</u>	[-]	отсутствие звука

Благодаря существованию фонем появляется возможность разделить акустическую модель на две части. Первая часть касается **произношения** и задает для каждого слова распределение вероятностей по возможным последовательностям фонем. Например, слово “ceiling” произносится как [s iy l ih ng]; или иногда как [s iy l ix ng], а иногда даже как [s iy l en]. Фонемы не являются непо-

средственно наблюдаемыми, поэтому, грубо говоря, речь может быть представлена как скрытая марковская модель, переменная состояния которой,  $X_t$ , определяет, какая фонема произносится в момент времени  $t$ .

Вторая часть акустической модели относится к тому способу, с помощью которого фонемы реализуются в виде акустических сигналов. Другими словами, переменная свидетельства  $E_t$  для скрытой марковской модели задает наблюдаемые характеристики акустического сигнала в момент времени  $t$ , а акустическая модель определяет вероятность  $P(E_t | X_t)$ , где  $X_t$  — текущая фонема. Эта модель позволяет также учитывать ударение, скорость и громкость речи и основана на методах из области **обработки сигналов**, позволяющих создавать описания сигналов, которые являются достаточно устойчивыми по отношению ко всем указанным влияниям.

В оставшейся части данного раздела приведено описание указанных моделей и алгоритмов, которое построено от нижнего уровня к верхнему, начиная от акустических сигналов и фонем, проходя через отдельные слова и заканчивая целыми предложениями. В заключение будет показано, как происходит обучение всех этих моделей и насколько хорошо работают результирующие системы.

## Звуки речи

Звуковые волны представляют собой периодические изменения давления, которые распространяются через воздух. Звук может быть измерен микрофоном, диафрагма которого смещается под воздействием изменений давления и вырабатывает непрерывно изменяющийся ток. Аналогово-цифровой преобразователь измеряет величину тока (которая соответствует текущей амплитуде звуковой волны) через дискретные интервалы, определяемые **частотой дискретизации**. Для обработки речи, как правило, применяется частота дискретизации от 8 до 16 кГц (т.е. от 8 до 16 тысяч раз в секунду). (Дискретизация высококачественных музыкальных записей осуществляется с частотой 44 кГц или больше.) Точность каждого измерения определяется **коэффициентом квантования**; в системах распознавания речи обычно применяется от 8 до 12 битов. Это означает, что в системах низкого класса дискретизация происходит с частотой 8 кГц и с квантованием 8 битами, а это требует для передачи фрагмента речи, занимающего одну минуту, примерно половины мегабайта. Было бы практически невозможно создавать и манипулировать распределениями вероятностей  $P(signal | phone)$  с таким большим объемом воспринимаемой информации, поэтому необходимо разработать более краткие описания акустического сигнала.

Прежде всего необходимо отметить следующее: хотя звуковые частоты в речи могут достигать нескольких килогерц, изменения в содержимом этого сигнала происходят гораздо менее часто, возможно, с частотой не больше 100 Гц. Поэтому в системах распознавания речи суммируются свойства сигнала за более продолжительные интервалы, называемые **фреймами**. Длина фрейма равна приблизительно 10 миллисекундам (т.е. соответствует 80 выборкам на частоте 8 кГц); это означает, что она достаточно мала, чтобы обеспечить исключение с помощью процесса суммирования некоторых помех, отличающихся меньшей продолжительностью. В пределах каждого фрейма происходящее в нем представляется с помощью вектора **акустических характеристик**. Например, во фрейме можно охарактеризовать количество энергии в каждом из нескольких частотных диапазонов. К другим важным характеристикам относится общее количество энергии во фрейме

и его отличие от предыдущего фрейма. Извлечение характеристик из речевого сигнала можно сравнить с прослушиванием выступления оркестра и определением того, что “теперь валторны звучат громко, а скрипки — тихо”. На рис. 15.14 показано, как происходят преобразования из непосредственно измеряемого звука в последовательность фреймов. Обратите внимание на то, что фреймы перекрываются; это позволяет предотвратить потерю информации, которая могла бы произойти, если бы важное акустическое событие случайно совпало с границей одного из фреймов.

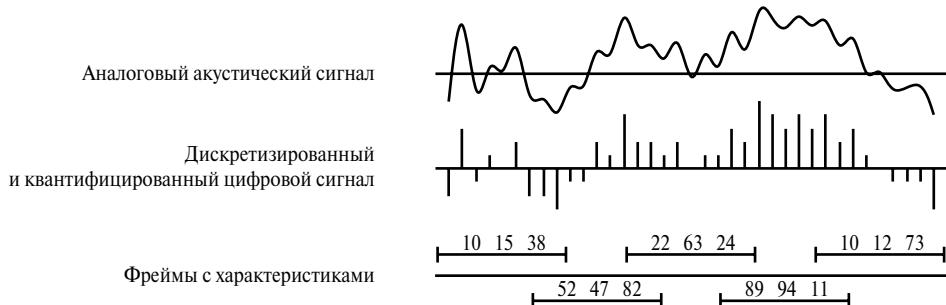


Рис. 15.14. Преобразование акустического сигнала в последовательность фреймов; для описания каждого фрейма применяются значения трех акустических характеристик

В данном случае показаны фреймы только с тремя характеристиками. В реальных системах используются десятки или даже сотни характеристик. Если применяется  $n$  характеристик и каждая из них имеет, скажем, 256 возможных значений, то любой фрейм представляется в виде точки в  $n$ -мерном пространстве и существует  $256^n$  возможных фреймов. При  $n > 2$  была бы практически неосуществимой попытка представить распределение вероятностей  $P(features|phone)$  в виде явно заданной таблицы, поэтому требуется дальнейшее сжатие. Ниже описаны два возможных подхода к решению этой задачи.

- В методе **векторного квантования**, или сокращенно VQ (Vector Quantization), все  $n$ -мерное пространство подразделяется, допустим, на 256 областей, обозначенных метками от C1 до C256. В таком случае появляется возможность представить каждый фрейм с помощью одной метки, а не вектора из  $n$  чисел. Поэтому в табулированном распределении  $P(VQ|phone)$  имеется 256 вероятностей, заданных для каждой фонемы. Но метод векторного квантования больше не находит широкого применения в крупномасштабных системах.
- Вместо дискретизации пространства характеристик для описания распределения  $P(features|phone)$  может использоваться параметризованное непрерывное распределение. Например, для каждой фонемы может применяться гауссово распределение с различными средними и матрицами ковариаций. Такой метод становится приемлемым, если акустические реализации каждой фонемы кластеризованы в отдельной области пространства характеристик. Но на практике звуки могут распределяться по некоторым областям, поэтому приходится использовать **сочетание гауссовых распределений**. Такое сочетание представляет собой взвешенную сумму  $k$  отдельных распределений, поэтому в распределении  $P(features|phone)$  имеется  $k$  весов,  $k$  векторов средних

с размером  $n$  и  $k$  матриц ковариации с размером  $n^2$ , т.е. для представления каждой фонемы применяется  $O(kn^2)$  параметров.

Очевидно, что при переходе от полного речевого сигнала к метке VQ или к множеству параметров сочетания распределений некоторая информация теряется. Весь секрет успешной обработки сигналов заключается в том, что характеристики и области (или гауссовы распределения) должны быть выбраны так, чтобы потери полезной информации свелись к минимуму. Любой конкретный звук речи может быть произнесен с помощью слишком многих способов: громко или тихо, быстро или медленно, с высоким или низким ударением, на фоне тишины или шума, а также любым из миллионов разных говорящих людей, каждый из которых имеет свой акцент и обладает разными характеристиками речевого тракта. Обработка сигналов должна осуществляться таким образом, чтобы были устранены все эти вариации и вместе с тем сохранилось то общее, чем характеризуется воспринимаемый звук<sup>5</sup>.

В простую модель, описанную выше, необходимо внести еще два уточнения. Первое из них относится к временной структуре фонем. При обычной речи большинство фонем имеет продолжительность 50–100 миллисекунд, т.е. фонемы занимают 5–10 фреймов. Для всех этих фреймов вероятностная модель  $P(\text{features} \mid \text{phone})$  является одинаковой, тогда как большинство фонем обладает ярко выраженной внутренней структурой. Например, фонема [t] представляет собой одну из нескольких **взрывных согласных**, при произнесении которых поток воздуха прерывается на короткое время, после чего резко освобождается. Изучая акустический сигнал, можно обнаружить, что фонема [t] имеет тихое начало, небольшой взрыв в середине и (обычно) шипение в конце. Эта внутренняя структура фонем может быть описана с помощью модели **фонемы с тремя состояниями**; каждая фонема имеет состояние *Onset* (Вступление), *Mid* (Середина) и *End* (Конец), а каждое состояние имеет свое собственное распределение среди вероятностей характеристик.

Второе уточнение касается контекста, в котором произносится фонема. Звучание каждой конкретной фонемы может изменяться под влиянием окружающих фонем<sup>6</sup>. Напомним, что звуки речи вырабатываются в результате движения губ, языка и нижней челюсти и проталкивания воздуха через голосовой тракт. Для координации этих сложных движений на скорости в пять или больше фонем в секунду мозг инициирует действия, относящиеся ко второй фонеме, еще до того, как оканчивается произнесение первой, что приводит к модификации одной или обеих фонем. Например, при произнесении слова “sweet” (сладкий) губы округляются еще во время произнесения фонемы [s] в предвидении того, что за ней последует фонема [w]. Такие **коартикуляционные эффекты** частично охватываются **трехфонемной** моделью, в которой в рамках акустической модели обеспечивается учет зависимости каждой фонемы от предшествующей и последующей фонем. Поэтому фонема [w] в слове “sweet” записывается как [w(s,iy)], т.е. как [w] с левым контекстом [s] и правым контекстом [iy].

<sup>5</sup> Противоположной задачей является задача **идентификации диктора**, в которой необходимо устраниТЬ общие характеристики и сохранить индивидуальные отличия, после чего попытаться сопоставить эти различия с моделями устной речи отдельных людей.

<sup>6</sup> Это означает, что “модель фонем” речи следует рассматривать, скорее, как полезную аппроксимацию, а не как незыблемый закон.

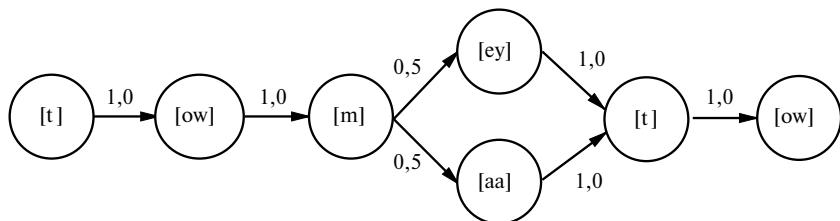
Результатом совместного применения модели трех состояний и трехфонемной модели становится увеличение количества возможных состояний временного процесса с  $n$  фонем первоначального фонетического алфавита ( $n \approx 50$  в случае ARPAbet) до  $3n^3$ . Но опыт показывает, что при этом достигается повышение точности, которое сторицей окупает дополнительные затраты на вероятностный вывод и обучение.

## Слова

Каждое слово можно рассматривать как определяющее отдельное распределение вероятностей  $\mathbf{P}(X_{1:t} | word)$ , где  $X_i$  задает состояние фонемы в  $i$ -м фрейме. Как правило, такое распределение делится на две части. **Модель произношения** задает распределение вероятностей по последовательностям фонем (игнорируя такие измерения, как время и состав фреймов), а **модель фонем** описывает то, как фонемы отображаются в последовательность фреймов.

Рассмотрим слово “tomato” (помидор). Согласно Гершвину [546], допустимыми являются варианты произношения этого слова  $[t \text{ ow } m \text{ ey } t \text{ ow}]$  и  $[t \text{ ow } m \text{ aa } t \text{ ow}]$ . На рис. 15.15, сверху показана модель перехода, в которой учитываются эти варианты. В данной модели имеются два возможных пути, один из которых соответствует последовательности фонем  $[t \text{ ow } m \text{ ey } t \text{ ow}]$ , а другой — последовательности  $[t \text{ ow } m \text{ aa } t \text{ ow}]$ . Вероятность любого из этих путей равна произведению вероятностей дуг, из которых состоит этот путь, как показано ниже.

a)



б)

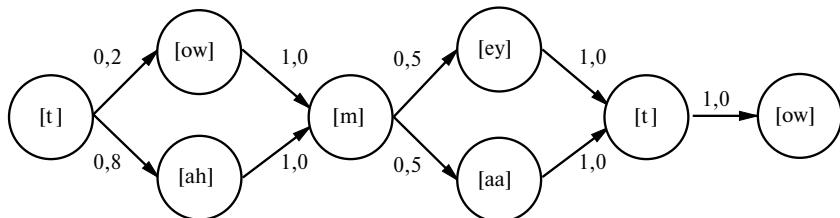


Рис. 15.15. Две модели произношения слова “томато”; каждая модель показана в виде диаграммы перехода с состояниями, обозначенными круглыми кружками, и допустимыми переходами, обозначенными стрелками, на которых показаны соответствующие вероятности: модель, допускающая учет различий между диалектами. Числовые оценки 0,5 основаны на том, что один из авторов данной книги предпочтет один из этих вариантов произношения, а другой автор предпочитает другой вариант (а); модель, в которой учитывается коартикуляционный эффект, возникающий при произнесении первой гласной; эта модель допускает наличие фонемы [ow] или [ah] (б)

$$P([\text{towmeytow}] \mid \text{"tomato"}) = P([\text{towmaatow}] \mid \text{"tomato"}) = 0.5$$

Вторым источником фонетических вариаций является **коартикуляция**. Например, фонема [t] формируется, когда язык находится в верхней части ротовой полости, а при произнесении фонемы [ow] язык должен находиться в нижней части. Во время быстрой речи язык часто оказывается в промежуточном положении и поэтому произносятся фонемы [t ah], а не [t ow]. На рис. 15.15, снизу приведена более сложная модель произношения слова "tomato", в которой принят в расчет этот коартикуляционный эффект. В данной модели имеются четыре отдельных пути, поэтому вероятности становятся таковыми:

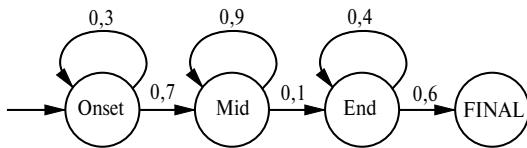
$$P([\text{towmeytow}] \mid \text{"tomato"}) = P([\text{towmaatow}] \mid \text{"tomato"}) = 0.1$$

$$P([\text{tahmeytow}] \mid \text{"tomato"}) = P([\text{tahmaatow}] \mid \text{"tomato"}) = 0.4$$

Аналогичные модели могут быть составлены для каждого слова, которое мы хотим распознать.

Модель для фонемы с тремя состояниями показана в виде диаграммы перехода между состояниями на рис. 15.16. Эта модель относится только к одной конкретной фонеме, [m], но все фонемы должны иметь модели с аналогичной топологией. Для каждого состояния фонемы показана связанная с ней акустическая модель, в которой принято предположение, что соответствующий акустический сигнал представлен меткой VQ. Например, согласно этой модели,  $P(E_t=C_1 \mid X_t=[m]_{\text{onset}}) = 0.5$ . Обратите внимание на то, что на данном рисунке показаны петли; например, состояние  $[m]_{\text{mid}}$  сохраняется с вероятностью 0.9, а это означает, что состояние  $[m]_{\text{mid}}$  имеет ожидаемую продолжительность 10 фреймов. В рассматриваемой модели продолжительность каждой фонемы является независимой от продолжительности других фонем; в более сложной модели могут проводиться различия между быстрой и медленной речью.

Скрытая марковская модель для фонемы [m]



Вероятности выходов для скрытой марковской модели этой фонемы

Onset:	Mid:	End:
$C_1: 0.5$	$C_3: 0.2$	$C_4: 0.1$
$C_2: 0.2$	$C_4: 0.7$	$C_6: 0.5$
$C_3: 0.3$	$C_5: 0.1$	$C_7: 0.4$

*Рис. 15.16. Скрытая марковская модель для фонемы [m] с тремя состояниями. Каждое состояние имеет несколько возможных выходов, каждый из которых обладает собственной вероятностью. Метки VQ от  $C_1$  до  $C_7$  выбраны произвольно, в качестве примера*

Аналогичные модели можно составить для каждой фонемы, возможно, с учетом трехфонемного контекста. Модель каждого слова, в сочетании с моделями его фо-

нем, задает полную спецификацию некоторой скрытой марковской модели, которая, в свою очередь, определяет вероятности перехода между состояниями фонем от фрейма к фрейму, а также вероятности акустических характеристик для каждого состояния фонем.

Если требуется распознавать  $\bowtie$  **отдельные слова** (т.е. слова, произнесенные без какого-либо окружающего контекста и с четкими границами), то необходимо найти слово, которое максимизирует следующее выражение:

$$P(\text{word} | e_{1:t}) = \alpha P(e_{1:t} | \text{word}) P(\text{word})$$

Априорную вероятность  $P(\text{word})$  можно получить по результатам обработки фактических речевых данных, а  $P(e_{1:t} | \text{word})$  представляет собой правдоподобие последовательности акустических характеристик, соответствующих модели рассматриваемого слова  $\text{word}$ . Вопросу о том, как вычисляются такие значения правдоподобия, посвящен раздел 15.2; в частности, в уравнении 15.5 определен простой метод рекурсивного вычисления, стоимость которого линейно зависит от  $t$  и от количества состояний марковской цепи. Чтобы найти наиболее вероятное слово, можно выполнить это вычисление для каждой возможной модели слова, умножить полученное значение на априорную вероятность и в соответствии с этим выбрать наиболее подходящее слово.

## Предложения

Для того чтобы поддерживать разговор с людьми, машина должна обладать способностью распознавать  $\bowtie$  **непрерывную речь**, а не просто отдельные слова. На первый взгляд может показаться, что непрерывная речь представляет собой не что иное, как последовательность слов, к которой вполне можно применить алгоритм, приведенный в предыдущем разделе. Но этот подход оканчивается неудачей по двум причинам. Прежде всего, как уже было показано (с. 732), последовательность наиболее вероятных слов не является наиболее вероятной последовательностью слов. Например, в кинофильме “Take the Money and Run” (Бери деньги и беги) банковский кассир прочитал каракули в записке героя Вуди Аллена как слова “I have a gub” (У меня есть штука). Хорошая языковая модель должна была бы предложить в качестве намного более вероятной последовательности слова “I have a gun” (У меня есть пушка), даже несмотря на то, что последнее слово больше похоже на “gub”, чем на “gun”. Вторая проблема, с которой приходится сталкиваться при обработке непрерывной речи, связана с  $\bowtie$  **сегментацией** — с проблемой определения того, где оканчивается одно слово и начинается следующее. С этой проблемой знаком любой, кто пытался изучать иностранный язык с помощью прослушивания устной речи, — на первых порах кажется, что все слова сливаются друг с другом. Но постепенно иностранец учится выделять отдельные слова из беспорядочных звуков. В данном случае первые впечатления вполне оправдываются; спектрографический анализ показывает, что в беглой речи слова действительно следуют одно за другим без пауз между ними. Поэтому нам приходится учиться определять границы между словами, несмотря на отсутствие пауз.

Начнем с языковой модели, назначение которой при распознавании речи состоит в определении вероятности каждой возможной последовательности слов. Используя запись  $w_1 \dots w_n$  для обозначения строки из  $n$  слов и  $w_i$  для обозначения  $i$ -го

слова в строке, можно составить выражение для вероятности некоторой строки с использованием цепного правила следующим образом<sup>7</sup>:

$$\begin{aligned} P(w_1 \dots w_n) &= P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2) \dots P(w_n | w_1 \dots w_{n-1}) \\ &= \prod_{i=1}^n P(w_i | w_1 \dots w_{i-1}) \end{aligned}$$

Большинство термов этого соотношения являются весьма сложными, а задача их оценки или вычисления является трудной. К счастью, возможно аппроксимировать эту формулу немного более простым соотношением и вместе с тем сохранить в целости значительную часть языковой модели. Одним из простых, широко применяемых и эффективных подходов является модель **дву словенных сочетаний**. В этой модели вероятность  $P(w_i | w_1 \dots w_{i-1})$  аппроксимируется вероятностью  $P(w_i | a_{i-1})$ . Иначе говоря, в этом подходе принимается предположение о том, что для последовательностей слов можно использовать марковскую цепь первого порядка.

Значительным преимуществом такой модели с двухсловными сочетаниями является то, что можно легко провести обучение этой модели, подсчитав, сколько раз каждая пара слов встречается в представительной совокупности строк, и используя эти подсчеты для оценки вероятности. Например, если “a” встречалось в обучающей совокупности 10 000, а за ним 37 раз следовало “gun”, то  $\hat{P}(gun_i | a_{i-1}) = 37 / 10,000$ , где под  $\hat{P}$  подразумевается оцениваемая вероятность. После такого обучения следует ожидать, что слова “I have” и “a gun” будут иметь высокие оцениваемые вероятности, а “I has” и “an gun” — низкие вероятности. В табл. 15.2 показаны некоторые результаты подсчета количества двухсловных сочетаний на примере слов в оригинал данной книги.

Возможно также перейти к использованию модели **трехсловных сочетаний**, в которой предусмотрены значения для  $P(w_i | w_{i-1} w_{i-2})$ . Это — более мощная языковая модель, позволяющая судить о том, что слова “ate a banana” (съесть банан) являются более вероятными, чем “ate a bandanna” (съесть бандану — цветной платок). Подходы, в которых предусмотрено применение моделей трехсловных сочетаний и в меньшей степени моделей двухсловных и однословных сочетаний, характеризуются наличием одной проблемы, которая связана с нулевыми результатами подсчета частоты: мы не должны утверждать, что какая-то комбинация слов невозможна, лишь потому, что она по стечению обстоятельств не встретилась в обучающей совокупности. Для того чтобы можно было назначить таким комбинациям небольшую ненулевую вероятность, применяется процесс **сглаживания**. Эта тема обсуждается на с. 1104.

Модели двух- или трехсловных сочетаний являются менее сложными по сравнению с некоторыми грамматическими моделями, которые будут рассматриваться в главах 22 и 23, но они позволяют лучше учесть локальные эффекты, связанные с контекстными зависимостями, и способны отразить определенные локальные синтаксические связи. Например, тот факт, что пары слов “I has” (я имеет) и “man

<sup>7</sup> Стого говоря, вероятность последовательности слов строго зависит от контекста фрагмента речи; например, слова “I have a gun” гораздо чаще встречаются в записках, передаваемых банковскому кассиру, чем, скажем, в статьях из журнала *Wall Street Journal*. Контекст учитывается лишь в немногих системах распознавания речи, а именно в тех, которые созданы путем изучения языковой модели специального назначения для решения конкретной задачи.

“have” (мужчина имею) получают низкие оценки, отражает общепринятые синтаксические соглашения по совместному использованию пар существительное–глагол. Проблема состоит в том, что подобные связи с помощью моделей сочетаний могут быть обнаружены только локально: неправильная языковая конструкция “the man have” (мужчина имею) получает низкую оценку, но более пространный оборот “the man with the yellow hat have” (мужчина в желтой шляпе имею) не рассматривается как ошибочный.

**Таблица 15.2.** Часть таблицы частот однословных и двухсловных сочетаний для слов в оригиналe данной книги. В ней наиболее часто применяемым отдельным словом является “the”, и общее количество случаев, в которых встречается это слово, равно 33 508 (из общего количества слов, равного 513 893). Наиболее часто встречающимся двухсловным сочетанием является “of the”, с общим количеством 3833. Некоторые частоты оказались больше ожидаемого (например, 4 раза встречается невероятное сочетание “on is”), поскольку при подсчете количества двухсловных сочетаний игнорируются знаки препинания, одно предложение может оканчиваться словом “он”, а другое — начинаться со слова “is”

Слово	Количество однословных сочетаний	Предыдущее слово							
		of	in	is	on	to	from	model	agent
the	33508	3833	2479	832	944	1365	597	28	24
on	2573	1	0	33	2	1	0	0	6
of	15474	0	0	29	1	0	0	88	7
to	11527	0	4	450	21	4	16	9	82
is	10566	3	6	1	4	2	1	47	127
model	752	8	1	0	1	14	0	6	4
agent	2100	10	3	3	2	3	0	0	36
idea	241	0	0	0	0	0	0	0	0

Теперь рассмотрим, как скомбинировать языковую модель со словесными моделями, чтобы иметь возможность правильно обрабатывать последовательности слов. Для упрощения предполагается, что будет использоваться двухсловная языковая модель. С помощью такой модели можно скомбинировать все словесные модели (которые, в свою очередь, состоят из моделей произношения и моделей фонем) в одну большую модель НММ. Состоянием в однословной модели НММ является фрейм с меткой, представляющей собой текущую фонему и состояние фонемы (например,  $[m]_{\text{Onset}}$ ); любое состояние в модели НММ непрерывной речи снабжается также меткой в виде слова, как, например,  $[m]_{\text{Onset}}^{\text{tomato}}$ . Если каждое слово в своей модели произношения имеет в среднем  $p$  фонем с тремя состояниями, а общее количество слов равно  $W$ , то модель НММ непрерывной речи имеет  $3pW$  состояний. Переходы могут происходить между состояниями фонем в пределах данной конкретной фонемы, между фонемами данного конкретного слова, а также между конечным состоянием одного слова и начальным состоянием другого. Переходы между словами происходят с вероятностями, заданными с помощью двухсловной модели.

После составления такой комбинированной модели НММ ее можно использовать для анализа непрерывного речевого сигнала. В частности, для обнаружения наиболее вероятной последовательности состояний может применяться алгоритм Виттерби, представленный в виде уравнения 15.9. Затем из этой последовательности

состояний можно извлечь последовательность слов, считывая метки слов из состояний. Таким образом, алгоритм Витерби позволяет решить проблему сегментации непрерывной речи на отдельные слова, поскольку в нем (по сути) используется динамическое программирование для одновременного учета не только всех возможных последовательностей слов, но и границ между словами.

Обратите внимание на то, что выше не было сказано, будто с помощью алгоритма Витерби “можно извлечь наиболее вероятную последовательность слов”. Дело в том, что наиболее вероятная последовательность слов не обязательно является такой, которая содержит наиболее вероятную последовательность состояний. Это связано с тем, что вероятность последовательности слов представляет собой сумму вероятностей по всем возможным последовательностям состояний, совместимых с данной последовательностью слов. Например, сравнивая две последовательности слов, скажем “a back” (спина) и “aback” (абак), можно обнаружить, что имеется десять альтернативных последовательностей состояний для “a back”, каждая из которых имеет вероятность 0,03, но только одна последовательность состояний для “aback” с вероятностью 0,20. Алгоритм Витерби выбирает “aback”, но фактически более вероятной является последовательность “a back”.

На практике это затруднение не исключает возможности применения данного подхода, но является достаточно серьезным для того, чтобы были предприняты попытки использовать другие подходы. Наиболее часто применяемым из них является алгоритм ~~декодера A\*~~ A\*, в котором предусмотрено остроумное использование поиска A\* (см. главу 4) для обнаружения наиболее вероятной последовательности слов. Идея этого алгоритма состоит в том, что каждая последовательность слов рассматривается как путь через граф, узлы которого обозначены метками в виде слов. Преемниками любого узла являются все слова, которые могут следовать за словом, являющимся меткой для этого узла; таким образом, граф для всех предложений с длиной, равной или меньшей  $n$ , имеет  $n$  уровней, причем каждый из этих уровней имеет максимальную ширину  $W$ , где  $W$  — количество возможных слов. При использовании двухсловной модели стоимость  $g(w_1, w_2)$  любой дуги между узлами с метками  $w_1$  и  $w_2$  задается выражением  $\log P(w_2 | w_1)$ ; таким образом, общая стоимость пути, соответствующего некоторой последовательности, может быть представлена следующим образом:

$$\text{Cost}(w_1 \dots w_n) = \sum_{i=1}^n -\log P(w_i | w_{i-1}) = -\log \prod_{i=1}^n P(w_i | w_{i-1})$$

При использовании такого определения стоимости пути задача поиска кратчайшего пути становится полностью эквивалентной задаче поиска наиболее вероятной последовательности слов. Для того чтобы этот процесс был достаточно эффективным, необходимо также иметь хорошую эвристику  $h(w_i)$  для оценки стоимости дополнения последовательности слов. Очевидно, что при этом подходе необходимо также учитывать, какая часть речевого сигнала еще не заменена словами из текущего пути. Тем не менее для решения этой задачи еще не были предложены эвристики, которые оказались бы особенно удачными.

## Разработка устройства распознавания речи

Качество системы распознавания речи зависит от качества всех ее компонентов — языковой модели, моделей произношения слов, моделей фонем и алгоритмов обработки сигналов, используемых для извлечения спектральных характеристик из акустического сигнала. Выше описано, как может быть составлена языковая модель, и указано, что для ознакомления с подробными сведениями об обработке сигналов следует обратиться к другим учебникам. Кроме того, в данной книге не рассматриваются модели произношения и модели фонем. Структура моделей произношения (таких как модели произношения слова “tomato”, показанные на рис. 15.15) обычно разрабатывается вручную. В настоящее время для английского языка и других языков составлены большие словари произношения, но далеко не все они отличаются приемлемой точностью. Структура моделей фонем с тремя состояниями является одинаковой для всех фонем, как показано на рис. 15.16. При использовании таких моделей остается только правильно определить сами вероятности. Как же можно получить такие данные, учитывая то, что для этих моделей могут потребоваться сотни тысяч или миллионы параметров?

Единственный осуществимый метод состоит в том, чтобы проводить обучение этих моделей по фактическим речевым данным, объем которых, безусловно, является буквально неограниченным. Очередная проблема заключается в том, как организовать такое обучение. Полный ответ на этот вопрос будет дан в главе 20, но в этом разделе мы можем изложить основные идеи. Рассмотрим двухсловную языковую модель; в данной главе описывалось, как провести ее обучение, подсчитывая частоты пар слов в реальном тексте. А можно ли применить такой же подход для определения вероятностей перехода между фонемами в модели произношения? Ответ на этот вопрос будет положительным, но только если кто-то возьмет на себя труд обозначить каждое вхождение каждого слова правильной последовательностью фонем. Это — трудная и чреватая ошибками задача, но она была выполнена для некоторых стандартных наборов данных, соответствующих нескольким часам речевых записей. Если известны последовательности фонем, то можно оценить вероятности перехода для моделей произношения на основе данных о частотах пар фонем. Аналогичным образом, если дано состояние фонем для каждого фрейма (а для получения этих данных требуется выполнить еще более трудоемкую работу по расстановке меток вручную), то можно оценить вероятности перехода для моделей фонем. Кроме того, если известны состояния и акустические характеристики фонем в каждом фрейме, то можно также оценить качество акустической модели либо непосредственно по данным о частотах (для моделей VQ), либо с использованием методов статистической подгонки (применительно к моделям, в которых применяется сочетание гауссовых распределений; см. главу 20).

Но указанный подход может не получить широкого распространения по таким причинам: данные с метками, проставленными вручную, обходятся дорого и встречаются редко, причем может оказаться, что даже доступные наборы данных с метками, расставленными вручную, не соответствуют тем типам говорящих людей и тем акустическим условиям, которые обнаруживаются в новом контексте распознавания речи.  К счастью, алгоритм ожидания-максимизации, или сокращенно алгоритм EM (Expectation Maximization), позволяет изучать модели перехода и модели восприятия HMM без необходимости использования данных с метками. Оценки, полученные на

основе данных с метками, расставленными вручную, могут использоваться для инициализации моделей; после этого управление берет на себя алгоритм EM и обеспечивает обучение моделей, предназначенных для решения предъявленной задачи. Идея функционирования этого алгоритма является простой: если дана некоторая модель HMM и последовательность наблюдений, то можно использовать алгоритмы сглаживания, описанные в разделах 15.2 и 15.3, для вычисления вероятности каждого состояния на каждом временном интервале, а затем, с помощью несложного дополнения, использовать его для вычисления вероятности каждой пары “состояние–состояние” на последовательных временных интервалах. Эти вероятности могут рассматриваться как неопределенные метки. С помощью этих неопределенных меток можно оценить новые вероятности перехода и восприятия, после чего повторить процедуру применения алгоритма EM. Такой метод гарантирует увеличение согласования между моделью и данными после каждой итерации и обычно сходится к гораздо более лучшему множеству значений параметров по сравнению с теми, которые были получены с помощью первоначальных оценок, сформированных по данным, размеченным вручную.

В современных системах распознавания речи используются колоссальные наборы данных и мощные вычислительные ресурсы для обучения применяемых в них моделей. В процессе распознавания отдельно сказанных слов в хороших акустических условиях (без фонового шума или реверберации) с помощью словаря из нескольких тысяч слов и при одном дикторе точность может превышать 99%. При распознавании неограниченной непрерывной речи с разными дикторами обычной является точность 60–80%, даже при хороших акустических условиях. А при наличии фонового шума и искажений, характерных для передачи речи по телефону, точность снижается еще больше. Хотя практически применимые системы совершенствовались в течение нескольких десятилетий, все еще остаются возможности для внедрения новых идей.

## 15.7. РЕЗЮМЕ

В настоящей главе рассматривалась общая проблема представления и формирования рассуждений о вероятностных временных процессах. Ниже перечислены основные идеи, изложенные в этой главе.

- Изменение состояния мира можно учесть, используя множество случайных переменных для представления этого состояния в каждый момент времени.
- Такие представления могут быть спроектированы таким образом, чтобы они удовлетворяли свойству **марковости**, согласно которому будущее не зависит от прошлого, если дано настоящее. В сочетании с предположением о том, что рассматриваемый процесс является **стационарным** (т.е. таким, что его законы не изменяются во временем), это позволяет намного упростить представление.
- Временная вероятностная модель может рассматриваться как содержащая **модель перехода**, которая описывает процесс развития, и **модель восприятия**, описывающая процесс наблюдения.
- Основными задачами вероятностного вывода во временных моделях являются **фильтрация, предсказание, сглаживание** и определение с помощью вычислений

**наиболее вероятного объяснения.** Каждая из этих задач может быть решена с использованием простых, рекурсивных алгоритмов, время выполнения которых линейно зависит от длины рассматриваемой последовательности.

- Немного более подробно были описаны три семейства временных моделей: **скрытые марковские модели, фильтры Калмана и динамические байесовские сети** (последняя модель включает две первых в качестве частных случаев).
- Двумя важными приложениями для временных вероятностных моделей являются **распознавание речи и слежение**.
- Если не приняты особые предположения, как при использовании фильтров Калмана, точный вероятностный вывод при наличии многих переменных состояния, по-видимому, становится неосуществимым. Создается впечатление, что на практике эффективным алгоритмом аппроксимации является алгоритм **фильтрации частиц**.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Многие важные идеи, касающиеся оценки состояния динамических систем, были высказаны математиком К.Ф. Гауссом [526], который сформулировал детерминированный алгоритм наименьших квадратов для решения задачи прогнозирования орбит небесных тел на основании астрономических наблюдений. Российский математик А.А. Марков [983] изложил в своих трудах, посвященных анализу стохастических процессов, подход, получивший в дальнейшем название **марковского предположения**; он провел оценку свойств марковской цепи первого порядка, состоящей из букв текста поэмы “Евгений Онегин”. Важная классификационная работа по фильтрации была выполнена во время Второй мировой войны Винером [1588] для непрерывных временных процессов и Колмогоровым [825] для дискретных временных процессов. Хотя эта научная деятельность привела к важным технологическим усовершенствованиям, достигнутым в течение следующих 20 лет, в ней использовалось представление на основе данных об области определения частот, поэтому многие вычисления оказались весьма громоздкими. Как и было указано Сверлингом [1482] и Калманом [764], непосредственное моделирование стохастических процессов с помощью пространства состояний оказалось намного проще. В последней статье предложен метод прямого вероятностного вывода в линейных системах с гауссовым шумом, который теперь известен под названием фильтров Калмана. Важные результаты в области сглаживания были получены Раухом и др. [1269], и метод, получивший выразительное название *метода сглаживания Рауха–Тунга–Стрибеля*, все еще широко применяется и в наши дни. Многие ранние результаты исследований были собраны в [531]. В [71] приведена более современная трактовка в байесовском стиле, а также многочисленные ссылки на необъятную литературу по этой теме. В [241] рассматривается “классический” подход к анализу временных рядов.

Во многих приложениях калмановской фильтрации приходится сталкиваться не только с неопределенными данными восприятия и неизвестными законами, но также и с неопределенной идентификацией; это означает, что если ведется текущий контроль за многочисленными объектами, система должна определить, какие данные наблюдений собраны от тех или иных объектов, прежде чем появится возмож-

ность обновить оценки состояний каждого из этих объектов. В этом заключается проблема ~~и ассоциирования данных~~ [70], [71]. При наличии  $n$  последовательностей наблюдений и  $n$  трактов слежения (т.е. в довольно благоприятном случае) существует  $n!$  возможных присваиваний последовательностей наблюдений трактам слежения; в правильной вероятностной трактовке должны учитываться все эти варианты присваивания, поэтому можно показать, что такая задача является NP-трудной [301], [302]. По-видимому, на практике хорошо работают методы аппроксимации с полиномиальными затратами времени, основанные на использовании алгоритма МСМС [1180]. Любопытно отметить, что задача ассоциирования данных представляет собой один из экземпляров задачи вероятностного вывода в языке первого порядка; в отличие от большинства задач вероятностного вывода, которые являются чисто пропозициональными, в задаче ассоциирования данных рассматриваются объекты, а также отношение идентификации. Поэтому она тесно связана с вероятностными языками первого порядка, которые упоминались в главе 14. В одной недавно опубликованной работе было показано, что формирование рассуждений об идентичности в общем и ассоциирование данных в частности могут осуществляться в рамках вероятностной инфраструктуры первого порядка [1179].

Скрытая марковская модель и связанные с ней алгоритмы вероятностного вывода и обучения, включая прямой–обратный алгоритм, были разработаны Баумом и Петри [85]. Аналогичные идеи были также высказаны независимо от них в сообществе специалистов по калмановской фильтрации [1269]. Прямой–обратный алгоритм был одним из основных предшественников более общей формулировки алгоритма EM [383]; см. также главу 20. Описание процедуры сглаживания в постоянном пространстве впервые появилось в [127], так же как и алгоритм, действующий по принципу “разделяй и властвуй”, который должен быть разработан при решении упр. 15.3.

Динамические байесовские сети (Dynamic Bayesian network — DBN) могут рассматриваться как способ разреженного кодирования марковского процесса; они были впервые применены в искусственном интеллекте Дином и Канадзава [361], Николсоном [1137] и Кьюрульфом [803]. Последняя работа включает описание общего дополнения к системе сетей доверия Hugin, которое предоставляет необходимые средства для формирования и компиляции динамической байесовской сети. Динамические байесовские сети нашли широкое применение для моделирования различных сложных процессов движения в системах машинного зрения [698], [718]. В [1443] явно показана связь между моделями НММ и сетями DBN, а также между прямым–обратным алгоритмом и алгоритмом распространения в байесовской сети. Результаты дальнейшего обобщения фильтров Калмана (и других статистических моделей) опубликованы в [1314].

Особенно интересной является история алгоритма фильтрации частиц, описанного в разделе 15.5. Первые алгоритмы формирования выборок для фильтрации были разработаны Хендшиным и Мейном [611], которые принадлежали к сообществу специалистов по теории управления, а идея повторного формирования выборок, лежащая в основе алгоритма фильтрации частицы, появилась в одной из публикаций в советском журнале по системам управления [1639]. В дальнейшем этот алгоритм был еще раз открыт в статистике и назван **последовательным повторным формированием выборок с учетом их важности**, или SIR (Sequential Importance-sampling Resampling) [939], [1315], в теории управления, под названием *фильтрация частиц* [581], [582], в искусственном интеллекте, под названием *выживание наиболее при-*

**способленного** [769], и в области машинного зрения, под названием **конденсация** [719]. Статья Канадзава и др. [769] включает одно усовершенствование, называемое **обращением свидетельства**, согласно которому выборка в состоянии на момент времени  $t+1$  осуществляется условно, в зависимости от состояния во время  $t$  и свидетельства во время  $t+1$ . Это позволяет обеспечить непосредственное влияние свидетельства на формирование выборок; в [406] было показано, что такой метод позволяет уменьшить ошибку аппроксимации.

Другие методы для аппроксимированной фильтрации включают алгоритм **вырожденной модели МСМС** [991] и метод факторизованной аппроксимации [164]. Оба метода обладают тем важным свойством, что ошибка аппроксимации не расходится во времени. Кроме того, для временных моделей были разработаны вариационные методы (см. главу 14). В [547] обсуждается алгоритм аппроксимации для **факторной модели НММ** — сети DBN, в которой две или несколько независимо развивающихся марковских цепей связаны с помощью разделяемого потока наблюдений. В [747] рассматривается целый ряд других приложений. Свойства продолжительностей смешивания обсуждаются в [959] и [1164].

Предыстория систем распознавания речи началась в 1920-х годах с создания игрушки Radio Rex — игрушечной собачки, активизируемой голосом. Собачка Rex прыгала в ответ на звуковые частоты около 500 Гц, которые соответствуют звучанию гласной [eɪ] в слове “Rex!”. Немного более серьезная работа в этой области началась после Второй мировой войны. В ATT Bell Labs была создана система для распознавания отдельно произносимых цифр [333] с помощью простого согласования акустических характеристик с шаблонами. Вероятности перехода между фонемами впервые использовались в системе, созданной в лондонском University College Фраем [508] и Денесом [384]. Начиная с 1971 года Агентство перспективных исследовательских программ (Defense Advanced Research Projects Agency — DARPA) Министерства обороны США финансировало четыре конкурирующих пятилетних проекта по разработке систем распознавания речи с высокой эффективностью. Победителем этого соревнования и единственной системой, соответствующей требованиям по распознаванию словаря из 1000 слов с точностью 90%, стала система Нагру<sup>8</sup>, разработанная в университете CMU [952], [953]. Окончательная версия системы Нагру была создана на основе системы Dragon, разработанной аспирантом СМУ Джеймсом Бейкером [62]; в системе Dragon впервые использовались скрытые марковские модели для распознавания речи. Почти одновременно с этим в компании IBM была разработана еще одна система на основе модели НММ [730]. Начиная с этого времени вероятностные методы в целом и скрытые марковские модели в частности стали доминировать в исследованиях и разработках по распознаванию речи. Последние годы характеризуются постепенным прогрессом, применением все более крупных наборов данных и моделей, а также ужесточением конкуренции в области решения все более реалистичных речевых задач. Некоторые исследователи изучали возмож-

<sup>8</sup> Система Hearsay-II [440], занявшая второе место в этом соревновании, испытала значительное влияние со стороны других направлений исследований в области искусственного интеллекта, поскольку в ней использовалась **архитектура классной доски**. Она представляла собой экспертную систему на основе правил с многочисленными, более или менее независимыми, модульными **источниками знаний**, взаимодействовавшими через общую **классную доску**, на которой они могли писать и читать. Системы классной доски стали основой современных архитектур пользовательского интерфейса.

ность использования сетей DBN вместо моделей НММ для распознавания речи с целью применения большей выразительной мощи сетей DBN для более полного охвата сложного скрытого состояния речевого аппарата [1286], [1652].

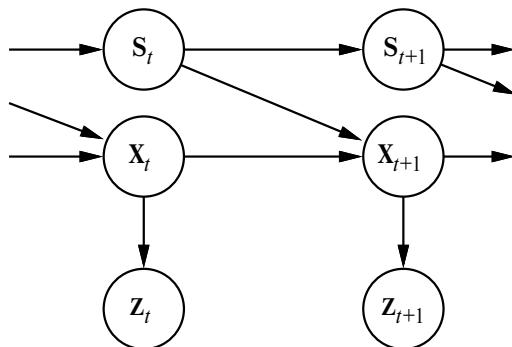
По проблематике распознавания речи имеется несколько хороших учебников: [568], [699], [731], [1263]. В [1550] собраны важные статьи в этой области, включая некоторые учебные руководства. Материал, представленный в данной главе, основан на обзоре, приведенном в [781], и на учебнике [756]. Результаты исследований в области распознавания речи публикуются в журналах *Computer Speech and Language*, *Speech Communications* и *IEEE Transactions on Acoustics, Speech, and Signal Processing*, в сборнике материалов семинаров *DARPA Workshops on Speech and Natural Language Processing*, а также в трудах конференций *Eurospeech*, *ICSLP* и *ASRU*.

## УПРАЖНЕНИЯ

---

- 15.1.** Покажите, что любой марковский процесс второго порядка может быть переформулен в виде марковского процесса первого порядка с дополненным множеством переменных состояния. Может ли такое преобразование всегда быть выполнено экономно, т.е. без увеличения количества параметров, необходимых для определения модели перехода?
- 15.2.** В этом упражнении рассматривается, что происходит с вероятностями в мире задачи с зонтиком по мере приближения к пределу в длинных временных последовательностях.
  - a) Предположим, что наблюдается нескончаемая последовательность дней, в которых директор появляется на работе с зонтиком или без зонтика. Покажите, что, по мере того, как проходят эти дни, вероятность дождя в текущий день возрастает монотонно в направлении к фиксированной точке. Рассчитайте эту фиксированную точку.
  - b) Теперь рассмотрим задачу прогнозирования все дальше и дальше в будущее по данным только первых двух наблюдений с зонтиком. Вначале рассчитайте вероятность  $P(R_{2+k} | U_1, U_2)$  для  $k=1 \dots 20$  и нанесите результаты на график. Вы должны обнаружить, что эта вероятность сходится в фиксированной точке. Рассчитайте точное значение для этой фиксированной точки.
- 15.3.** В данном упражнении разрабатывается вариант прямого–обратного алгоритма, приведенного в листинге 15.1, характеризующийся эффективным использованием пространства. Требуется вычислить значение  $P(\mathbf{x}_k | \mathbf{e}_{1:t})$  для  $k=1, \dots, t$ . Такую задачу можно решить с помощью подхода по принципу “разделяй и властвуй”.
  - a) Для упрощения примем предположение, что значение  $t$  является нечетным, и допустим, что промежуточная точка определяется выражением  $h = (t+1)/2$ . Покажите, что значение  $P(\mathbf{x}_k | \mathbf{e}_{1:t})$  можно вычислить для  $k=1, \dots, h$ , если даны лишь первоначальное прямое сообщение  $\mathbf{f}_{1:0}$ , обратное сообщение  $\mathbf{b}_{h+1:t}$  и свидетельство  $\mathbf{e}_{1:h}$ .
  - b) Покажите аналогичный результат для второй половины последовательности.

- в) Если даны результаты выполнения заданий 15.3, а и 15.3, б, рекурсивный алгоритм “разделяй и властвуй” можно сформировать, вначале выполнив прогон в прямом направлении вдоль последовательности, а затем в обратном направлении от ее конца, сохранив лишь необходимые сообщения в середине и в концах. Затем алгоритм вызывается на каждой половине последовательности. Составьте подробный листинг этого алгоритма.
- г) Определите временную и пространственную сложность алгоритма как функцию от  $t$ , длины последовательности. Как изменятся эти результаты, если входные данные будут разделены больше чем на две части?
- 15.4.** На с. 732 была кратко описана ошибочная процедура определения наиболее вероятной последовательности состояний, в которой используется последовательность наблюдений. В этой процедуре предусматривается поиск в каждом временном интервале наиболее вероятного состояния, применение операции сглаживания и возврат последовательности, в которой собраны эти состояния. Покажите, что при использовании некоторых временных вероятностных моделей и последовательностей наблюдений эта процедура возвращает невозможную последовательность состояний (т.е. такую последовательность, что ее апостериорная вероятность равна нулю).
- 15.5.** Часто возникает необходимость осуществлять текущий контроль за системой с непрерывным состоянием, поведение которой переключается непредсказуемым образом с одного режима на другой в множестве из  $k$  различных режимов. Например, самолет, пытающийся избежать поражения ракетой, может выполнить ряд различных маневров, которые попытается отследить система управления ракетой. Представление такой модели **переключательного фильтра Калмана** в виде байесовской сети показано на рис. 15.17.



*Рис. 15.17. Представление переключательного фильтра Калмана в виде байесовской сети. Переключательная переменная  $S_t$  представляет собой дискретную переменную состояния, значение которой определяет модель перехода для непрерывных переменных состояния  $\mathbf{x}_t$ . Для любого дискретного состояния  $i$  модель перехода  $P(\mathbf{x}_{t+1} | \mathbf{x}_t, S_t=i)$  представляет собой линейную гауссову модель, так же как и в обычном фильтре Калмана. Модель перехода для дискретного состояния,  $P(S_{t+1} | S_t)$ , может рассматриваться как матрица, по аналогии со скрытой марковской моделью*

- a) Допустим, что дискретное состояние  $S_t$  имеет  $k$  возможных значений и что априорная непрерывная оценка состояния  $P(\mathbf{x}_0)$  представляет собой многомерное гауссово распределение. Покажите, что предсказание  $P(\mathbf{x}_1)$  представляет собой **сочетание гауссовых распределений**, т.е. такую взвешенную сумму гауссовых распределений, что веса в сумме составляют 1.
- б) Покажите, что если текущая оценка непрерывного состояния  $P(\mathbf{x}_t | \mathbf{e}_{1:t})$  представляет собой сочетание  $m$  гауссовых распределений, то в общем случае обновленная оценка состояния  $P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1})$  будет представлять собой сочетание  $km$  гауссовых распределений.
- в) Какой аспект временного процесса представляют веса в сочетании гауссовых распределений?

Результаты выполнения заданий 15.5, *а* и 15.5, *б*, вместе взятые, показывают, что объем этого представления апостериорных вероятностей беспрепятственно возрастает даже при использовании переключательных фильтров Калмана, которые являются простейшими гибридными динамическими моделями.

- 15.6.** Дополните недостающий этап вывода уравнения 15.17 — первый этап обновления для одномерного фильтра Калмана.
- 15.7.** Рассмотрим ход выполнения операции обновления дисперсии в уравнении 15.18.
- а) Нанесите на график значения выражения  $\sigma_t^2$  как функции от  $t$  при наличии различных значений для  $\sigma_x^2$  и  $\sigma_z^2$ .
- б) Покажите, что эта операция обновления имеет фиксированную точку, такую, что  $\sigma^2 \sigma_t^2 \rightarrow \sigma^2$ , по мере того как  $t \rightarrow \infty$ , и рассчитайте значение  $\sigma^2$ .
- в) Дайте качественное объяснение того, что происходит по мере того, как  $\sigma_x^2 \rightarrow 0$  и  $\sigma_z^2 \rightarrow 0$ .
- 15.8.** Покажите, как представить модель НММ в виде рекурсивной реляционной вероятностной модели в соответствии с предложением, приведенным в разделе 14.6.
- 15.9.** В этом упражнении более подробно анализируется устойчивая к отказам модель для датчика аккумулятора, показанная на рис. 15.11, *а*.
- а) График на рис. 15.11, *б* обрывается при  $t=32$ . Дайте качественное описание того, что должно произойти по мере стремления  $t$  к бесконечности,  $t \rightarrow \infty$ , если датчик продолжает выдавать показания 0.
- б) Предположим, что температура окружающей среды влияет на датчик аккумулятора таким образом, что по мере возрастания температуры временные отказы становятся все более вероятными. Покажите, как дополнить с учетом этого структуру сети DBN, приведенную на рис. 15.11, *а*, и объясните, какие изменения потребуется внести в таблицы условных вероятностей.
- в) После определения новой структуры этой сети, может ли робот использовать показания датчика аккумулятора для вероятностного вывода данных о текущей температуре?
- 15.10.** Рассмотрите задачу применения алгоритма устранения переменной к сети DBN для задачи с зонтиком, развернутую на три временных среза, в которой

используется запрос  $P(R_3 | U_1, U_2, U_3)$ . Покажите, что сложность этого алгоритма (размер наибольшего фактора) является одинаковой, независимо от того, устраниются ли переменные, касающиеся дождя, в прямом или обратном порядке.

- 15.11.** В модели произношения слова “tomato”, приведенной на рис. 15.15, допускается коартикуляция с первой гласной, поскольку даны две возможные фонемы. Альтернативный подход состоит в использовании трехфонемной модели, в которой фонема  $[ow(t, m)]$  автоматически включает изменение в гласном звуке. Составьте полную трехфонемную модель для слова “tomato”, включая вариант, касающийся диалекта.
- 15.12.** Вычислите наиболее вероятный путь через модель НММ, приведенную на рис. 15.16, для выходной последовательности  $[C_1, C_2, C_3, C_4, C_4, C_6, C_7]$ . Кроме того, приведите значение его вероятности.

# 16 ПРИНЯТИЕ ПРОСТЫХ РЕШЕНИЙ

*В этой главе показано, как должен принимать решения агент, чтобы получать то, что ему требуется, по крайней мере в среднем количестве случаев.*

В данной главе мы вернемся к идеи теории полезности, которая была представлена в главе 13, и покажем, как можно объединить эту теорию с теорией вероятностей, чтобы создать агента, действующего на основе теории решений, т.е. агента, способного принимать рациональные решения с учетом того, в чем он убежден и к чему стремится. Такой агент может принимать решения в таких ситуациях, когда из-за неопределенности и конфликтующих целей логический агент остается неспособным что-либо решить. По существу, агент, основанный на цели, может лишь проводить бинарное различие между хорошими (целевыми) и плохими (нечелевыми) состояниями, тогда как агент, основанный на теории решений, пользуется непрерывными показателями качества состояния.

В разделе 16.1 изложен основной принцип теории решений — максимизация ожидаемой полезности. В разделе 16.2 показано, что поведение любого рационального агента можно понять, определив функцию полезности, которую он максимизирует. В разделе 16.3 более подробно обсуждаются характерные особенности функций полезности и, в частности, то, как они связаны с отдельными величинами, такими как деньги. В разделе 16.4 показано, как обращаться с функциями полезности, которые зависят от нескольких величин. В разделе 16.5 описана реализация систем принятия решений. В частности, в этом разделе описан формальный подход, называемый **сетями принятия решений** (известный также под названием **диаграмм влияния**); такие сети являются дополнением байесовских сетей, в которое включены действия и показатели полезности. В оставшейся части этой главы рассматриваются проблемы, связанные с применением теории решений в экспертных системах.

## 16.1. СОВМЕСТНЫЙ УЧЕТ УБЕЖДЕНИЙ И ЖЕЛАНИЙ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ

В книге *Port-Royal Logic* (“Логика Пор-Ройяль”), написанной в 1662 году, французский философ Арно изложил следующую мысль.

“Чтобы судить о том, что следует делать, чтобы получить хорошее или избежать плохого, необходимо рассматривать не только хорошее и плохое само по себе, но и вероятность того, произойдет ли оно или не произойдет, а также рассматривать математически пропорцию, в которой все эти обстоятельства встречаются вместе.”

В современной литературе вместо понятий *хорошего* или *плохого* применяется понятие *полезности*, но общий принцип остается таким же. Для описания предпочтений агента, с помощью которых он различает состояния мира, применяется **функция полезности**, которая присваивает состоянию единственное числовое значение, чтобы показать, насколько оно желательно. Полезности объединяются с вероятностями действий для определения ожидаемой полезности каждого действия.

В этой главе будет использоваться запись  $U(S)$  для обозначения полезности состояния  $S$  с точки зрения агента, принимающего решения. На данный момент мы будем рассматривать состояния как полные снимки параметров мира, по аналогии с **ситуациями**, которые рассматривались в главе 10. На первых порах такой подход позволяет упростить изложение данной темы, но когда придется отдельно определять полезность каждого возможного состояния, он может стать довольно громоздким. В разделе 16.4 будет показано, как можно выполнить в некоторых обстоятельствах декомпозицию состояний в целях присваивания значений полезности.

Любое недетерминированное действие  $A$  имеет возможные результирующие состояния  $Result_i(A)$ , где индекс  $i$  пробегает по различным результатам. Прежде чем осуществить действие  $A$ , агент присваивает вероятность  $P(Result_i(A) | Do(A), E)$  каждому результату, где  $E$  представляет собой сумму доступных агенту свидетельств о мире, а  $Do(A)$  — высказывание, согласно которому действие  $A$  выполняется в текущем состоянии. Таким образом, можно вычислить **ожидаемую полезность** действия с учетом свидетельства,  $EU(A | E)$ , с использованием следующей формулы:

$$EU(A | E) = \sum_i P(Result_i(A) | Do(A), E) U(Result_i(A)) \quad (16.1)$$

Принцип **максимальной ожидаемой полезности** (Maximum Expected Utility — MEU) гласит, что рациональный агент должен выбирать действие, которое максимизирует ожидаемую полезность для агента. А если бы с помощью этого уравнения потребовалось выбрать наилучшую последовательность действий, то пришлось бы перенумеровать все последовательности действий и выбрать наилучшую; очевидно, что такой подход при наличии длинных последовательностей действий становится неосуществимым. Поэтому данная глава посвящена главным образом описанию простых решений (которые обычно предусматривают единственное действие), а в следующей главе будут представлены новые методы, позволяющие эффективно применять последовательности действий.

В определенном смысле принцип MEU может рассматриваться как определение всего искусственного интеллекта. Все, что должен делать интеллектуальный агент, сводится к вычислению различных количественных величин, определению максимальной полезности по всем своим действиям, а затем осуществлению этих действий. Но сказанное не означает, что тем самым проблема искусственного интеллекта решена по определению!

Хотя принцип MEU позволяет определить правильное действие, которое должно быть выполнено в любой задаче принятия решений, связанный с этим объем вычис-

лений может оказаться неосуществимым, а иногда нелегко даже полностью сформулировать саму задачу. Для того чтобы определить начальное состояние мира, требуется применить восприятие, обучение, представление знаний, логический и вероятностный вывод. Для вычисления вероятностей  $P(\text{Result}_i(A) | \text{Do}(A), E)$  необходимо иметь полную причинную модель мира, а также, как было показано в главе 14, осуществлять NP-трудный вероятностный вывод в байесовских сетях. Для вычисления полезности каждого состояния,  $U(\text{Result}_i(A))$ , часто требуется поиск или планирование, поскольку агент не может определить, насколько хорошим является состояние, до тех пор, пока не узнает, чего он может достичь из этого состояния. Поэтому теория принятия решений — это не панацея, которая позволила бы решить всю проблему искусственного интеллекта. С другой стороны, она предоставляет инфраструктуру, с помощью которой можно определить, где должны найти свое место те или иные компоненты любой системы искусственного интеллекта.

Очевидно, что принцип MEU связан с идеей показателей производительности, которая была представлена в главе 2. Эта основная идея очень проста. Рассмотрим варианты среды, действия в которых могут привести к получению агентом данной конкретной истории восприятий, а также предположим, что существует возможность спроектировать несколько разных агентов. *Если агент максимизирует функцию полезности, правильно отражающую показатели производительности, по которым можно судить о его поведении, то этот агент достигнет наивысших возможных значений показателей производительности, если будет проведено усреднение полученных значений показателей по всем вариантам среды, в которые может быть помещен этот агент.* Это определение представляет собой также основное обоснование для самого принципа MEU. Хотя на первый взгляд такое определение может показаться содержащим тавтологию, фактически оно воплощает в себе очень важный переход от глобального, внешнего критерия рациональности (по результатам оценки производительности на основании историй восприятия в среде) к локальному, внутреннему критерию, основанному на максимизации функции полезности применительно к следующему состоянию.

В данной главе будут рассматриваться только единственные, или **единоразовые решения**, тогда как в главе 2 были определены показатели производительности, измеряемые по историям восприятия в среде, которые обычно становятся результатом многих решений. В следующей главе, посвященной описанию **последовательных решений**, будет показано, как можно согласовать эти два подхода к оценке функционирования агента.

## 16.2. ОСНОВЫ ТЕОРИИ ПОЛЕЗНОСТИ

Интуитивно ясно, что принцип максимальной ожидаемой полезности (Maximum Expected Utility — MEU) может стать основой приемлемого способа принятия решений, но нет никаких оснований полагать, что это — единственный рациональный способ. В конечном итоге, на каком основании следует придавать такое значение подходу, в котором предусматривается максимизация средней полезности? Почему бы не попытаться максимизировать сумму кубов возможных полезностей или не рассмотреть подход с минимизацией наихудшей возможной потери? Кроме того, может ли агент рационально организовать свои действия, лишь выражив отношение предпочтения между состояниями и не присваивая им числовых значений? Нако-

нец, почему вообще должна существовать функция полезности с требуемыми свойствами? Вполне допустимо такое предположение, что рациональный агент может иметь структуру предпочтений, являющуюся слишком сложной для того, чтобы ее можно было представить с помощью такого простого метода, как присваивание единственного действительного числового значения каждому состоянию.

### Ограничения, налагаемые на рациональные предпочтения

Ответы на эти вопросы можно получить, записав некоторые ограничения, распространяющиеся на предпочтения, которые должен иметь рациональный агент, а затем показав, что принцип MEU можно вывести из этих ограничений. Для описания предпочтений агента будут использоваться приведенные ниже обозначения.

$A \succ B$ . Вариант  $A$  предпочтительнее, чем  $B$

$A \sim B$ . Агент безразличен к выбору между вариантами  $A$  и  $B$

$A \succsim B$ . Агент предпочитает вариант  $A$  варианту  $B$  или безразличен к выбору между ними

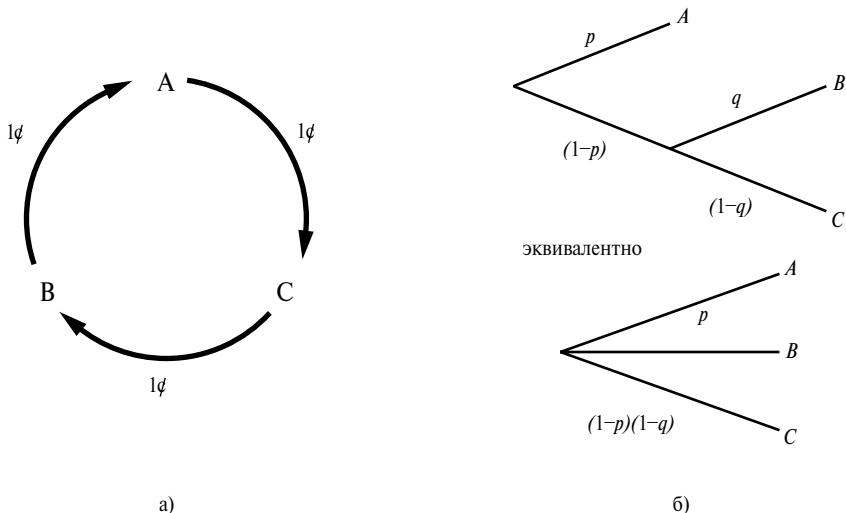
Теперь напрашивается очевидный вопрос, к какого рода понятиям относятся  $A$  и  $B$ ? Если действия агента являются детерминированными, то  $A$  и  $B$  обычно представляют собой конкретные, полностью заданные результирующие состояния этих действий. В более общем, недетерминированном случае  $A$  и  $B$  представляют собой **лотореи**. Лоторея по сути является распределением вероятностей по множеству фактических результатов (“призов” в лотерее). Лоторея  $L$  с возможными результатами  $C_1, \dots, C_n$ , которые могут возникать с вероятностями  $p_1, \dots, p_n$ , записывается следующим образом:

$L = [p_1, C_1; p_2, C_2; \dots; p_n, C_n]$

(Лоторея только с одним результатом может быть записана либо как  $A$ , либо как  $[1, A]$ .) Вообще говоря, каждым результатом лотореи может быть атомарное состояние или другая лоторея. Основная проблема теории полезности состоит в том, чтобы понять, как предпочтения между сложными лотореями связаны с предпочтениями между основополагающими состояниями в этих лотореях.

Для этого мы должны наложить на это отношение предпочтения приемлемые ограничения, во многом аналогично тому, как были наложены ограничения рациональности на степени уверенности для получения аксиом вероятностей в главе 13. Одно из разумных ограничений состоит в том, что предпочтение должно быть **транзитивным**, т.е. если  $A \succ B$  и  $B \succ C$ , то следует ожидать, что  $A \succ C$ . Свойство транзитивности можно обосновать, показав, что агент, предпочтения которого не соответствуют свойству транзитивности, будет вести себя нерационально. Предложим, например, что агент имеет нетранзитивное предпочтение  $A \succ B \succ C \succ A$ , где  $A$ ,  $B$  и  $C$  — товары, которые могут свободно обмениваться друг на друга. Если агент в настоящее время имеет товар  $A$ , то можно предложить обменять  $C$  на  $A$  и немногих наличных. Агент предпочитает товар  $C$  и поэтому должен быть готов предоставить определенную сумму денег, чтобы заключить эту сделку. Затем можно предложить обменять  $B$  на  $C$ , добавив еще немногих денег, и, наконец, обменять  $A$  на  $B$ . После этого мы вернемся к такому же положению, с которого начинали, за исклю-

чением того, что теперь агент имеет меньше денег (рис. 16.1, *a*). Такое движение по циклу может продолжаться до тех пор, пока у агента больше не останется денег. Очевидно, что в этом случае агент не действовал рационально.



*Рис. 16.1. Пример нарушения свойства транзитивности: циклическое повторение обменов товаром, которое показывает, что нетранзитивные предпочтения  $A \succ B \succ C \succ A$  приводят к нерациональному поведению (а); графическая иллюстрация аксиомы декомпонуемости (б)*

Приведенные ниже шесть ограничений известны как аксиомы теории полезности. Они определяют самые очевидные семантические ограничения, которые распространяются на предпочтения и лотереи.

- **Упорядочиваемость.** Если даны два состояния, то рациональный агент должен либо предпочесть одно другому, либо рассматривать их оба как в равной степени предпочтительные. Это означает, что агент не может избежать принятия решений. Как было сказано на с. 637, отказ сделать ставку аналогичен тому, что агент отказывается предпринимать активные действия и предоставляет все дальнейшее естественному течению событий.

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

- **Транзитивность.** Если даны три состояния, такие, что агент предпочитает состояние *A* состоянию *B* и предпочитает состояние *B* состоянию *C*, то агент должен предпочесть состояние *A* состоянию *C*.

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

- **Непрерывность.** Если некоторое состояние *B* находится в порядке предпочтений между *A* и *C*, то существует некоторая вероятность *p* того, что рациональный агент будет безразличен к тому, чтобы определенно выбрать *B* или лотерею, результатом которой является состояние *A*, с вероятностью *p*, и состояние *C*, с вероятностью  $1-p$ .

$$A \succ B \succ C \Rightarrow \exists p [p, A; 1-p, C] \sim B$$

- **Заменяемость.** Если агент безразличен к выбору между двумя лотереями,  $A$  и  $B$ , то агент безразличен и к выбору между двумя более сложными лотереями, которые являются одинаковыми, за исключением того, что в одной из них подставлена лотерея  $B$  вместо  $A$ . Такое свойство сохраняется независимо от вероятностей и от других результатов лотерей.

$$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$$

- **Монотонность.** Предположим, что две лотереи имеют два одинаковых результата,  $A$  и  $B$ . Если агент предпочитает состояние  $A$  состоянию  $B$ , то агент должен предпочесть лотерею, которая имеет более высокую вероятность для состояния  $A$  (и наоборот).

$$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \succsim [q, A; 1-q, B])$$

- **Декомпонуемость.** Сложные лотереи можно свести к простым, используя законы вероятностей. Это свойство получило название правила “экономии количества ставок”, поскольку согласно ему две последовательные лотереи могут быть сжаты в одну эквивалентную лотерею<sup>1</sup> (см. рис. 16.1, б).

$$[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$$

## В начале была Полезность

Обратите внимание на то, что в аксиомах теории полезности ничего не сказано о самой полезности; в них речь идет только о предпочтениях. Предполагается, что основным свойством рациональных агентов является умение пользоваться предпочтениями, а существование функции полезности следует из аксиом полезности, как показано ниже.

### 1. Принцип полезности

Если предпочтения агента подчиняются аксиомам полезности, то существует функция  $U$  с реальными значениями, областью определения которой являются состояния, такая, что  $U(A) > U(B)$  тогда и только тогда, когда  $A$  является более предпочтительным, чем  $B$ , и  $U(A) = U(B)$  тогда и только тогда, когда агент безразличен к выбору между  $A$  и  $B$ .

$$U(A) > U(B) \Leftrightarrow A \succ B$$

$$U(A) = U(B) \Leftrightarrow A \sim B$$

### 2. Принцип максимальной ожидаемой полезности

Полезность лотереи равна сумме произведений вероятности каждого результата на полезность этого результата.

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

---

<sup>1</sup> Мы можем учсть саму привлекательность игры на деньги, включив события игры на деньги в описание состояния; например, действие “взять с собой 10 долларов и сделать ставку” может рассматриваться как более предпочтительное, чем “взять с собой 10 долларов и не сделать ставку”.

Другими словами, как только будут определены вероятности и полезности всех возможных результирующих состояний, полезность сложной лотереи, охватывающей эти состояния, становится полностью определенной. Поскольку результатом недетерминированного действия является лотерея, существует возможность вывести это правило принятия решений на основании принципа МЕУ из уравнения 16.1.

Важно помнить, что из существования функции полезности, которая описывает поведение агента по выбору предпочтений, не обязательно следует, что агент явно максимизирует эту функцию полезности в своих собственных размышлениях. Как было показано в главе 2, рациональное поведение может быть выработано с помощью самых различных способов, причем некоторые из них являются более эффективными по сравнению с явной максимизацией полезности. Но наблюдение за предпочтениями рационального агента дает возможность составить функцию полезности, которая служит представлением того, что фактически пытается достичь агент своими действиями.

### 16.3. ФУНКЦИИ ПОЛЕЗНОСТИ

*Полезность* — это функция, которая отображает состояния на действительные числа. Исчерпывается ли на этом все, что можно сказать о функциях полезности? Строго говоря, так оно и есть. С учетом соблюдения ограничений, перечисленных выше, агент может иметь любые предпочтения, какие пожелает. Например, агент может предпочесть держать на своем банковском счете такое количество долларов, которое измеряется простым числом; в этом случае, если ему на счет поступило 16 долларов, он может растратить 3 доллара. Агент вправе предпочесть “зубастый” Ford Pinto выпуска 1973 года сверкающему новому автомобилю Mercedes. Кроме того, предпочтения могут взаимодействовать друг с другом; например, агент вправе предпочитать иметь количество долларов, выражаящихся простыми числами, когда он владеет автомобилем Ford Pinto, а когда в его собственности находится Mercedes, он может предпочитать иметь больше долларов, чем меньше.

Но если бы все функции полезности были такими же произвольными, как эта, то теория полезности не могла бы оказать нам значительную помощь, поскольку пришлось бы понаблюдать за предпочтениями агента во всех возможных сочетаниях обстоятельств, прежде чем получить возможность делать какие-либо предсказания в отношении его поведения. К счастью, предпочтения реальных агентов обычно в большей степени укладываются в определенную систему. И наоборот, существуют систематические способы проектирования функций полезности, которые после установки их в искусственном агенте вынуждают этого агента вырабатывать варианты поведения таких типов, какие нам требуются.

#### Полезность денег

Корни теории полезности скрываются в экономике, а экономика предоставляет одного очевидного кандидата для использования в качестве меры полезности — деньги (или, более конкретно, общий суммарный капитал агента). Почти универсальная способность денег к обмену на всевозможные товары и услуги подсказывает, что деньги играют важную роль в функциях полезности людей. (В действительно-

сти большинство людей рассматривают экономику как науку о деньгах, тогда как фактически по своему происхождению слово **экономия** относится к управлению хозяйством, а современное направление экономических исследований нацелено на обоснование рационального выбора.)

Если мы ограничим свое внимание только такими действиями, которые влияют на количество денег, имеющихся у агента, то, как правило, обнаружим, что при всех прочих равных условиях агент предпочитает иметь больше денег, а не меньше. Таким образом, можно утверждать, что агент проявляет склонность к **монотонному предпочтению** применительно к определенным суммам денег. Однако этого недостаточно, чтобы можно было использовать деньги в качестве значения функции полезности, поскольку в этом определении ничего не сказано о предпочтениях между лотериями, связанными с денежными ставками.

Представьте себе, что вы одержали победу над всеми соперниками в телевизионном игровом шоу, и ведущий предлагает вам выбор: забирайте свой приз в 1 000 000 долларов или сделайте на него ставку, бросив монету. Если выпадет орел, вы ничего не получите, а если выпадет решка, то получите 3 000 000 долларов. Если вы — такой же, как большинство людей, то откажетесь от этой ставки и положите в карман миллион. Является ли это решение нерациональным?

При условии, что вы уверены в подлинности этой монеты, **ожидаемое денежное значение** (Expected Monetary Value — EMV) этой ставки равно

$$\frac{1}{2}(0 \text{ долларов}) + \frac{1}{2}(3 000 000 \text{ долларов}) = 1 500 000 \text{ долларов},$$

а значение EMV взятия первоначального приза, безусловно, равно 1 000 000 долларов, т.е. меньше. Но такой расчет не обязательно означает, что принятие предложения сделать эту ставку является лучшим решением. Предположим, что мы используем запись  $S_n$  для обозначения состояния, соответствующего обладанию всей суммой  $n$  долларов, а ваши текущие накопления составляют  $k$  долл. В таком случае ожидаемые полезности двух действий, соответствующих принятию предложения сделать ставку (*Accept*) и отказу от него (*Decline*), выражаются следующими соотношениями:

$$\begin{aligned} EU(\text{Accept}) &= \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+3\ 000\ 000}) \\ EU(\text{Decline}) &= U(S_{k+1\ 000\ 000}) \end{aligned}$$

Чтобы определить, что делать, необходимо присвоить значения полезности результатившим состояниям. Полезность не является прямо пропорциональной денежному значению, поскольку полезность вашего первого миллиона (связанная с положительным изменением образа жизни) очень высока (по крайней мере, все так говорят), тогда как полезность еще одного миллиона гораздо меньше. Предположим, что присвоено значение полезности 5 текущему финансовому состоянию ( $S_k$ ), 10 — состоянию  $S_{k+3\ 000\ 000}$  и 8 — состоянию  $S_{k+1\ 000\ 000}$ . В таком случае рациональное действие должно состоять в том, чтобы отказаться от предложения сделать ставку, поскольку ожидаемая полезность его принятия равна только 7.5 (меньше 8, что соответствует отказу от этого предложения). С другой стороны, предположим, что некто уже имеет 500 000 000 долларов на банковском счете (и, вполне можно предположить, участвует в игровых шоу только ради развлечения). В таком случае указанное предложение, по-видимому, является вполне приемлемым,

так как дополнительная польза от появления 503-го миллиона, скорее всего, почти не отличается от той, которая соответствует получению 501-го миллиона.

В своем оригинальном исследовании фактически применяемых функций полезности Грейсон [589] обнаружил, что полезность денег почти точно пропорциональна логарифму их количества (предположение об этом впервые высказал Бернулли [111]; см. упр. 16.3). Одна конкретная кривая, относящаяся к данным о предпочтениях некоего мистера Берда, показана на рис. 16.2, а. Полученные Грейсоном данные о предпочтениях мистера Берда совместимы со следующей функцией полезности для диапазона от  $n = -150\ 000$  долларов до  $n = 800\ 000$  долларов:

$$U(S_{k+n}) = -263.31 + 22.09 \log(n+150\ 000)$$

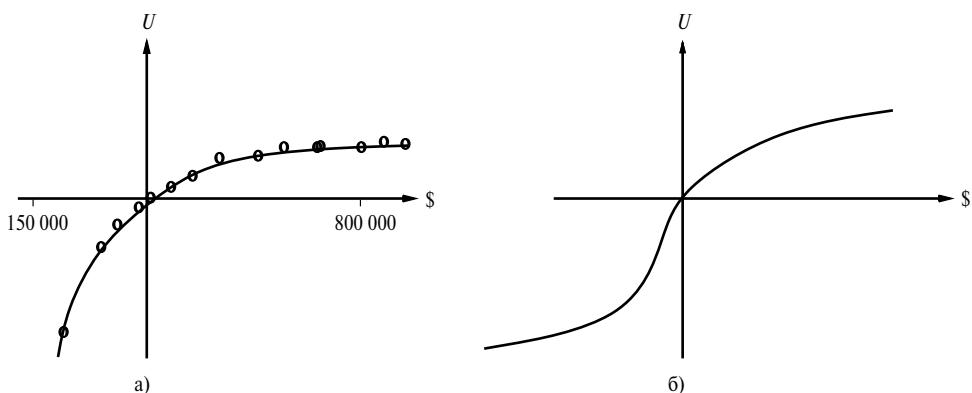


Рис. 16.2. Кривая полезности денег: эмпирические данные о предпочтениях мистера Берда, измеренные в ограниченном диапазоне (а); типичная кривая для полного диапазона (б)

Не следует считать, что это — безусловно верная функция полезности для денежных значений, но создается впечатление, что большинство людей руководствуются функцией полезности, которая является вогнутой в области положительных значений денежных накоплений. Брать в долг обычно считается катастрофическим решением, но предпочтения между различными уровнями задолженности могут показывать обратное поведение по отношению к вогнутости, связанной с положительными денежными накоплениями. Например, если некто уже имеет долг 10 000 000 долларов, то вполне может принять предложение сделать ставку на подбрасывание подлинной монеты, с выигрышем в 10 000 000 долларов после выпадения орла и проигрышем в 20 000 000 долларов после выпадения решки<sup>2</sup>. Такое поведение соответствует \$-образной\$ кривой, показанной на рис. 16.2, б.

Если мы ограничим наше внимание положительной частью таких кривых, где уклон постепенно уменьшается, то для любой лотереи  $L$  полезность решения, при котором придется столкнуться с выбором в этой лотерее, меньше, чем полезность получения ожидаемого денежного значения в этой лотерее без всяких условий:

$$U(L) < U(S_{EMV(L)})$$

<sup>2</sup> Такое поведение можно назвать отчаянным, но оно рационально, если человек уже находится в безнадежной ситуации.

Это означает, что агенты с кривыми полезности такой формы  $\curvearrowleft$  **избегают риска**: они предпочитают надежное приобретение, пусть даже с меньшей отдачей по сравнению с ожидаемым денежным значением возможной ставки. С другой стороны, в “безнадежной” области с большими отрицательными накоплениями на рис. 16.2, б поведение характеризуется  $\curvearrowleft$  **стремлением к риску**. Сумма, приобретаемая агентом вместо лотереи, называется  $\curvearrowleft$  **эквивалентом определенности** лотереи. Исследования показали, что большинство людей предпочитают 400 долларов вместо ставки, розыгрыш которой позволяет получить в половине случаев 1000 долларов, а в другой половине случаев получить 0 долларов; это означает, что эквивалентом определенности для этой лотереи является 400 долларов. Разность между ожидаемым денежным значением лотереи и ее эквивалентом определенности называется  $\curvearrowleft$  **страховой премией**. Неприятие риска является основой индустрии страхования, поскольку такое поведение означает, что страховые премии становятся положительными. Люди предпочитают заплатить небольшую страховую премию, нежели делать ставку на всю стоимость своего дома против шанса потерять его в огне. А с точки зрения страховой компании цена отдельного дома весьма мала по сравнению с общими резервами этой компании. Это означает, что кривая полезности страховщика в таком небольшом регионе остается приблизительно линейной и для этой компании ставка на стоимость дома против страховой премии является беспроигрышной.

Обратите внимание на то, что при небольших изменениях в уровне денежных накоплений по сравнению с текущими накоплениями почти любая кривая полезности должна быть приблизительно линейной. Агент, который руководствуется такой линейной кривой, называется  $\curvearrowleft$  **нейтрально относящимся к риску**. Поэтому в случае ставок на небольшие суммы предполагается соблюдение свойства нейтрального отношения к риску. В определенном смысле это свойство является обоснованием упрощенной процедуры, в которой применяются небольшие ставки для оценки вероятностей, а также обоснованием аксиом вероятностей, приведенных в главе 13.

### СУБЪЕКТИВНЫЕ СУЖДЕНИЯ И ПРИСУЩЕЕ ЧЕЛОВЕКУ СВОЙСТВО ОШИБАТЬСЯ

Теория принятия решений является  $\curvearrowleft$  **нормативной** — она описывает, как должен действовать рациональный агент. Область применения экономической теории удалось бы значительно расширить, если бы существовала также какая-то  $\curvearrowleft$  **описательная** теория фактического принятия решений людьми. Однако имеются экспериментальные свидетельства, которые показывают, что люди систематически нарушают аксиомы теории полезности. Один из примеров такого поведения приведен психологами Тверским и Канеманом [1523], который основан на примере экономиста Алле [14]. Участникам проводимого им эксперимента был предоставлен выбор между лотереями A и B, а затем между лотереями C и D:

- A: 80% шансов на получение 4000 долларов
- B: 100% шансов на получение 3000 долларов
- C: 20% шансов на получение 4000 долларов
- D: 25% шансов на получение 3000 долларов

Большинство участников предпочли B перед A и C перед D. Но если присвоить значению полезности нулевой суммы нулевое значение,  $U(0 \text{ долларов}) = 0$ , то

из первого из этих выборов следует, что  $0.8U(4000 \text{ долларов}) < U(3000 \text{ долларов})$ , а из второго следует прямо противоположное. Другими словами, создается впечатление, что не существует функции полезности, которая была бы совместимой с этими выборами. Одно из возможных заключений состоит в том, что люди действуют нерационально с точки зрения стандартов, определяемых приведенными выше аксиомами полезности. Но в одном из альтернативных подходов к трактовке этого явления предполагается, что в этом анализе не учитывается **сожаление** — чувство, которое, как известно людям, они будут испытывать, если откажутся от надежного приобретения (B) в обмен на 80%-ные шансы более высокого вознаграждения, а затем проиграют. Иначе говоря, если будет выбрана лотерея A, то перед людьми будет стоять 20%-ный шанс не получить никаких денег и почувствовать себя полным идиотом.

Канеман и Тверский приступили к разработке описательной теории, которая объясняет, почему люди избегают риска в условиях высокой вероятности выигрыша, но готовы делать более рискованные ставки при менее вероятном вознаграждении. Связь между этими исследованиями и искусственным интеллектом состоит в том, что выборы, сделанные нашими агентами, являются лишь настолько качественными, насколько являются таковыми предпочтения, на которых они основаны. Если же люди, информирующие нас о своих взглядах, настаивают на своих противоречивых суждениях о предпочтениях, то искусственный агент не может ничего сделать, чтобы получить возможность поступать в дальнейшем так же, как они.

К счастью, суждения людей, касающиеся предпочтений, часто бывают открыты для пересмотра в свете дополнительных размышлений. В своей ранней работе по оценке полезности денег, выполненной в Гарвардской школе бизнеса, Кини и Райффа [788] изложили описанные ниже результаты.

- “Значительный объем проведенных эмпирических исследований показал серьезный недостаток применяемого нами протокола оценки. Участники экспериментов проявляли слишком большую склонность избегать риска в малом и поэтому... составленные на основании экспериментов функции полезности показывают недопустимо большие премии за риск для лотерей с большим разбросом значений выигрыша. ...Но большинство участников эксперимента смогли понять, что их действия не обоснованы, и испытывали такие чувства, будто ими освоен важный урок в отношении того, как им следует себя вести. Сделав соответствующие выводы, некоторые участники эксперимента отказались от страховки против автомобильной аварии и заключили договора страхования своей жизни на более продолжительный срок.”

Даже в наши дни продолжаются интенсивные исследования в области рациональности (нерациональности) поведения людей.

## Шкалы полезности и оценка полезности

Если даже известно, каково поведение агента в отношении предпочтений, аксиомы полезности не позволяют определить для него уникальную функцию полезности. Например, любую функцию полезности  $U(S)$  можно преобразовать в другую функцию:

$$U'(S) = k_1 + k_2 U(S)$$

где  $k_1$  — некоторая константа;  $k_2$  — любая положительная константа. Очевидно, что в результате этого линейного преобразования поведение агента останется неизменным.

В детерминированных контекстах, где имеются состояния, но не лотереи, поведение не изменяется под действием любого монотонного преобразования. Например, можно вычислить квадратный корень всех значений полезности, не повлияв на упорядочение действий по предпочтениям. Принято считать, что агент в детерминированной среде руководствуется **функцией значения**, или **порядковой функцией полезности**; по существу, эта функция обеспечивает лишь ранжирование состояний и не позволяет получить осмыслиенные числовые значения. Это различие было показано в главе 6 для игр: функции оценки в таких детерминированных играх, как шахматы, представляют собой функции значения, тогда как функции оценки в недетерминированных играх, подобных нардам, являются истинными функциями полезности.

Одна из процедур оценки полезностей состоит в том, что определяется шкала с “наилучшим возможным выигрышем” при  $U(S) = u_T$  и “наихудшим возможным проигрышем” при  $U(S) = u_L$ . Для **нормализованных значений полезности** используется шкала с  $u_L=0$  и  $u_T=1$ . Оценка полезностей промежуточных результатов осуществляется путем опроса агента, который должен обозначить свое предпочтение между заданным результирующим состоянием  $S$  и **стандартной лотереей**  $[p, u_T; (1-p), u_L]$ . Вероятность  $p$  корректируется до тех пор, пока агент не становится безразличным к выбору между  $S$  и этой стандартной лотереей. Это означает, что при наличии нормализованных полезностей полезность  $S$  определяется вероятностью  $p$ .

В задачах принятия решений, касающихся медицинского обслуживания, транспорта и охраны окружающей среды, кроме всего прочего, ставками становятся жизни людей. В таких случаях  $u_L$  представляет собой значение, присвоенное немедленной смерти (или, возможно, многим смертям). *Хотя все испытывают смущение, когда назначается цена человеческой жизни, истина состоит в том, что во всех критических ситуациях приходится применять компромиссы с учетом этой цены.* Самолеты проходят полный цикл технического обслуживания через интервалы, определяемые количеством полетов и длиной пробега в километрах, а не после каждого полета. Корпуса автомобилей изготавливаются из относительно тонкого листового металла для уменьшения расходов, несмотря на то, что это приводит к уменьшению относительного количества людей, выживших после аварии. Этилированное топливо продолжает широко использоваться, даже несмотря на то, что всем известно, какой вред оно наносит здоровью. Как ни парадоксально, отказ “назначить цену жизни в деньгах” означает, что жизнь часто недооценивается. Росс Шахтер рассказывал о своем опыте общения с одним правительственный агентством, которое заказало исследование по вопросу целесообразности удаления асбестовых изделий из школ. Шахтер провел исследование, в котором было принято предположение, что цена жизни ребенка школьного возраста имеет конкретное долларовое выражение, и доказал, что при таком предположении рациональным выбором должно стать удаление асбестовых изделий. Правительственное агентство, ссылаясь на соображения морали, даже не стало рассматривать отчет по результатам этого исследования, после чего было принято решение о том, что не следует заниматься удалением асбестовых изделий.

Было сделано несколько попыток определить, какую цену назначают люди за свои собственные жизни. Двумя “валютами”, широко используемыми в медицинских и страховых исследованиях, являются **микрошанс смерти** (*micromort* — один из мил-

лиона шансов смерти) и  $\Delta$  QALY (Quality-Adjusted Life Year), или год жизни с поправкой на качество жизни (который эквивалентен году, прожитому в добром здравии, без каких-либо заболеваний). Целый ряд исследований, проведенных с охватом самых разных слоев населения, показал, что микрошанс смерти стоит примерно 20 долларов (в долларах по курсу за 1980 год). Как уже было показано, функции полезности не обязательно должны быть линейными, поэтому из полученных результатов не следует, что некое лицо, принимающее решение, согласилось бы подвергнуться смерти за 20 миллионов долларов. К тому же локальная линейность любой кривой полезности означает, что показатели микрошанса смерти и QALY являются наиболее приемлемыми для анализа небольших инкрементных рисков и вознаграждений.

## 16.4. МНОГОАТРИБУТНЫЕ ФУНКЦИИ ПОЛЕЗНОСТИ

Принятие решений в области общественной политики, с одной стороны, связано с затратами в миллионы долларов, а с другой стороны, часто касается вопросов жизни и смерти. Например, принимая решение о том, какие уровни канцерогенного вещества являются допустимыми в окружающей среде, лица, устанавливающие нормы допустимых загрязнений, вынуждены искать компромисс между требованиями по предотвращению преждевременных смертей и экономическими трудностями, которые могут возникнуть из-за отказа от некоторых промышленных продуктов и процессов. При поиске строительной площадки для нового аэропорта приходится учитывать, какой вред окружающей среде будет нанесен этим строительством, стоимость земельного участка, расстояние от центров сосредоточения большого количества населения, шум, связанный с деятельностью аэропорта, проблемы безопасности, обусловленные местной топографией и погодными условиями и т.д. Задачи, подобные этим, в которых результаты характеризуются двумя или несколькими атрибутами, рассматриваются в  $\Delta$  теории многоатрибутивной полезности.

Обозначим эти атрибуты как  $\mathbf{x} = \langle X_1, \dots, X_n \rangle$ ; полный вектор присваиваний принимает вид  $\mathbf{x} = \langle x_1, \dots, x_n \rangle$ . Вообще говоря, предполагается, что каждый атрибут имеет дискретные или непрерывные скалярные значения. Для упрощения принимается допущение, будто каждый атрибут определен таким образом, что при всех прочих равных условиях более высокие значения атрибута соответствуют большим полезностям. Например, если в качестве одного из атрибутов в задаче с аэропортом будет выбрано отсутствие шума *AbsenceOfNoise*, то решение будет тем лучше, чем выше значение этого атрибута. В некоторых случаях может потребоваться подразделить диапазон значений так, чтобы значение полезности изменялось монотонно в каждом поддиапазоне.

Начнем с анализа случаев, в которых решения могут быть приняты без комбинирования значений атрибутов в одно значение полезности. Затем рассмотрим случаи, в которых полезности комбинаций атрибутов могут быть определены очень кратко.

### Доминирование

Предположим, что площадка для аэропорта  $S_1$  стоит меньше, способствует выработке меньшего шумового загрязнения и характеризуется большей безопасностью, чем площадка  $S_2$ . В таком случае следует без колебаний отвергнуть вариант с пло-

щадкой  $S_2$ . В этой ситуации применяется такая формулировка, что имеет место **строгое доминирование** варианта  $S_1$  над вариантом  $S_2$ . Вообще говоря, если некоторый вариант характеризуется меньшими значениями всех атрибутов по сравнению с каким-то другим вариантом, то нет необходимости продолжать его дальнейшее рассмотрение. Отношение строгого доминирования часто бывает очень полезным, когда требуется сократить перечень вариантов выбора, оставив лишь реальных претендентов, хотя его применение редко приводит к тому, что остается лишь один уникальный вариант. На рис. 16.3, *a* показана диаграмма для случая с двумя атрибутами.

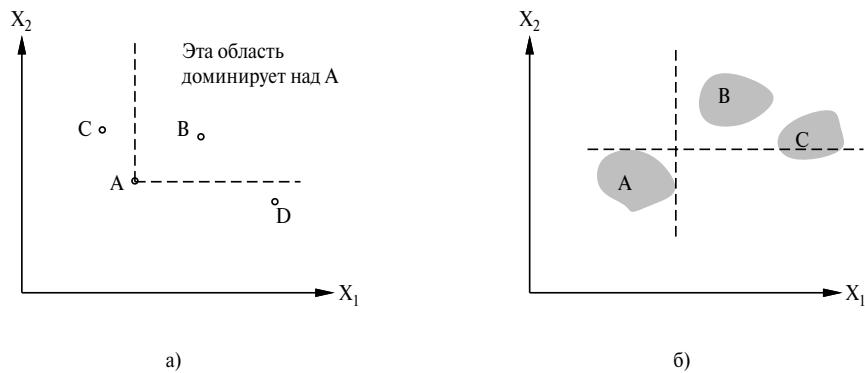


Рис. 16.3. Страгое доминирование: детерминированный случай; над вариантом А строго доминирует вариант В, но не варианты С или Д (*а*); неопределенный случай; над вариантом А строго доминирует вариант В, но не вариант С (*б*)

Такой подход является вполне применимым для детерминированного случая, в котором точно известны значения атрибутов. А как следует поступать в общем случае, когда результаты действий являются неопределенными? И в этих обстоятельствах может применяться непосредственный аналог отношения строгого доминирования, в котором, несмотря на неопределенность, все возможные конкретные результаты для  $S_1$  строго доминируют над всеми возможными результатами для  $S_2$  (см. рис. 16.3, *б*). Безусловно, что такая ситуация, вполне вероятно, может возникать даже реже по сравнению с детерминированным случаем.

К счастью, возможно более полезное обобщение, называемое **стохастическим доминированием**, которое встречается очень часто в реальных задачах. Принцип стохастического доминирования проще понять в контексте задачи с одним атрибутом. Допустим, исследования показали, что стоимость строительства аэропорта на площадке  $S_1$  равномерно распределена в пределах от 2,8 миллиарда долларов до 4,8 миллиарда долларов, а стоимость строительства на площадке  $S_2$  равномерно распределена в пределах от 3 миллиардов долларов до 5,2 миллиарда долларов. На рис. 16.4, *а* показаны эти распределения в виде графиков, на которых указанные значения отображены в виде отрицательных значений. В таком случае при наличии лишь той информации, что полезность уменьшается с увеличением стоимости, можно прийти к заключению, что вариант  $S_1$  стохастически доминирует над вариантом  $S_2$  (и поэтому вариант  $S_2$  может быть отброшен). Важно отметить, что такой вывод не следует из сравнения ожидаемых затрат. Например, если бы было известно,

что стоимость варианта  $S_1$  точно равна 3,8 миллиарда долларов, то мы не смогли бы принять решение без дополнительной информации о полезности денег<sup>3</sup>.

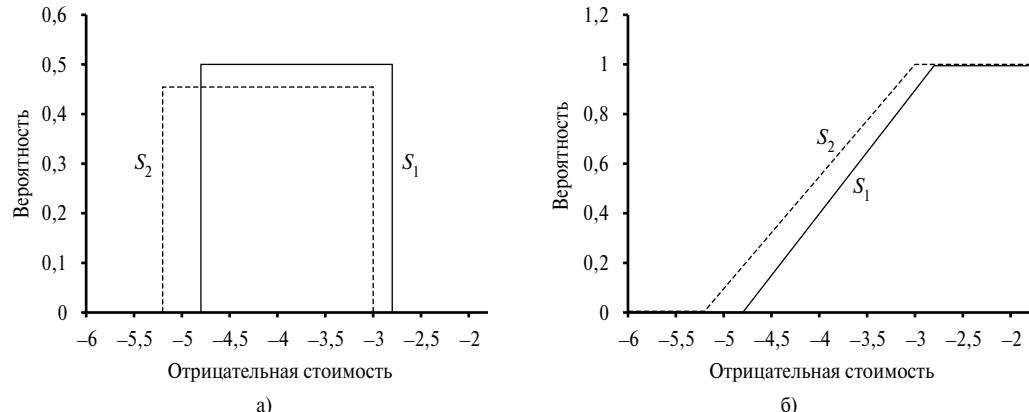


Рис. 16.4. Стохастическое доминирование: вариант  $S_1$  стохастически доминирует над вариантом  $S_2$  по стоимости (а); кумулятивные распределения для отрицательной стоимости вариантов  $S_1$  и  $S_2$  (б)

Точные соотношения между распределениями стоимостей атрибутов, необходимые для определения стохастического доминирования, можно проще всего оценить, исследуя кумулятивные распределения, показанные на рис. 16.4, б. В кумулятивном распределении измеряется вероятность того, что стоимость меньше или равна какой-либо заданной сумме, т.е. в этом распределении интегрируется первоначальное распределение. Если кумулятивное распределение для  $S_1$  всегда находится справа от кумулятивного распределения для  $S_2$ , то с точки зрения стохастической оценки вариант  $S_1$  дешевле, чем  $S_2$ . С формальной точки зрения, если два действия,  $A_1$  и  $A_2$ , приводят к созданию распределений вероятностей  $p_1(x)$  и  $p_2(x)$  по атрибуту  $X$ , то действие  $A_1$  стохастически доминирует над действием  $A_2$  по атрибуту  $X$ , если справедливо следующее соотношение:

$$\forall x \int_{-\infty}^x p_1(x') dx' \leq \int_{-\infty}^x p_2(x') dx'$$

Возможность применения этого определения для выбора оптимальных решений вытекает из следующего свойства: если действие  $A_1$  стохастически доминирует над действием  $A_2$ , то для любой неубывающей монотонно функции полезности  $U(x)$  ожидаемая полезность действия  $A_1$  является, по меньшей мере, столь же высокой, как и ожидаемая полезность действия  $A_2$ . Если какое-то действие стохастически доминирует над другим действием по всем атрибутам, то последнее действие должно быть отброшено.

<sup>3</sup> На первый взгляд может показаться странным, что увеличение объема информации о стоимости варианта  $S_1$  может уменьшить способность агента к принятию решений. Этот парадокс можно разрешить, отметив, что решение, достигнутое в отсутствии точной информации о стоимости, с меньшей вероятностью будет иметь максимальную отдачу с точки зрения полезности.

На первый взгляд может показаться, что условие стохастического доминирования является довольно формальным и, возможно, не позволяющим проводить вычисления без трудоемких вероятностных расчетов. Но в действительности оно во многих случаях позволяет легко принимать решения. Предположим, например, что стоимость строительства зависит от расстояний до центров сосредоточения населения. Сама стоимость остается неопределенной, но чем больше указанные расстояния, тем выше стоимость. Если площадка  $S_1$  менее удалена, чем  $S_2$ , то вариант  $S_1$  будет доминировать над  $S_2$  по стоимости. Хотя в данной книге эта тема не рассматривается, существуют точные алгоритмы распространения качественной информации такого рода среди неопределенных переменных в **качественных вероятностных сетях**, которые позволяют системе вырабатывать рациональные решения на основе отношений стохастического доминирования, без использования каких-либо числовых значений.

### Структура предпочтений и многоатрибутная полезность

Предположим, что имеются  $n$  атрибутов, каждый из которых имеет  $d$  различных возможных значений. Чтобы определить полную функцию полезности  $U(x_1, \dots, x_n)$ , в худшем случае требуется  $d^n$  значений. Итак, наихудший случай соответствует ситуации, в которой предпочтения агента вообще не отличаются какой-либо регулярностью. Теория многоатрибутной полезности основана на гипотезе о том, что предпочтения типичных агентов более структурированы по сравнению с указанной ситуацией. Основной подход состоит в том, что следует выявлять регулярные структуры в том поведении агента по отношению к предпочтениям, которое, по всей вероятности, будет наблюдаться в действительности, и использовать так называемые **теоремы представления** для обоснования того, что агент со структурой предпочтений определенного рода имеет следующую функцию полезности:

$$U(x_1, \dots, x_n) = f[f_1(x_1), \dots, f_n(x_n)]$$

где  $f$ , в соответствии с оптимистическим предположением, представляет собой простую функцию, такую как сложение. Обратите внимание на то, что попытка определить структуру предпочтений аналогична использованию байесовских сетей для декомпозиции совместного распределения вероятностей нескольких случайных переменных.

#### Предпочтение без неопределенности

Начнем с детерминированного случая. Напомним, что для детерминированных вариантов среды у агента имеется функция значений  $V(x_1, \dots, x_n)$ ; цель состоит в том, чтобы представить эту функцию в более краткой форме. Основное свойство регулярности, которое наблюдается в детерминированных структурах предпочтений, называется **независимостью предпочтений**. Два атрибута,  $X_1$  и  $X_2$ , являются независимыми по предпочтениям от третьего атрибута,  $X_3$ , если предпочтение между результатами  $\langle x_1, x_2, x_3 \rangle$  и  $\langle x_1', x_2', x_3 \rangle$  не зависит от конкретного значения  $x_3$  для атрибута  $X_3$ .

Возвращаясь к примеру с аэропортом, в котором нужно было рассмотреть (кроме других атрибутов) атрибуты *Noise* (Шум), *Cost* (Стоимость) и *Deaths* (Количество смертных случаев), можно предположить, что атрибуты *Noise* и *Cost* независимы

по предпочтениям от атрибута *Deaths*. Например, если мы предпочтим состояние с 20 000 людей, проживающих в районах, над которыми выполняются полеты, и стоимостью строительства 4 миллиарда долларов, состоянию с 70 000 людей, проживающих в районах полетов, и стоимостью 3,7 миллиарда долларов, притом что уровень безопасности в обоих случаях равен 0,06 смертей в расчете на миллионный пассажирооборот, то будем иметь одни и те же предпочтения, когда уровень безопасности равен 0,13 и когда он равен 0,01. Такие же отношения независимости имеют место для предпочтений между любыми другими парами значений атрибутов *Noise* и *Cost*. Очевидно также, что атрибуты *Cost* и *Deaths* независимы по предпочтениям от *Noise*, а *Noise* и *Deaths* независимы по предпочтениям от *Cost*. В этих случаях принято считать, что множество атрибутов  $\{Noise, Cost, Deaths\}$  обнаруживает свойство **взаимной независимости по предпочтениям** (Mutual Preferential Independence — MPI). Согласно свойству MPI, вне зависимости от того, насколько важен каждый атрибут, он не влияет на отношения, в которых другие атрибуты сопоставляются друг с другом.

Свойство взаимной независимости по предпочтениям в определенной степени представляет собой идеализацию, но благодаря замечательной теореме, предложенной экономистом Дебре [365], на его основе можно вывести очень простую форму для функции стоимости агента: **если атрибуты  $X_1, \dots, X_n$  являются взаимно независимыми по предпочтениям, то поведение агента в отношении его предпочтений можно описать как максимизацию следующей функции:**

$$V(x_1, \dots, x_n) = \sum_i V_i(x_i)$$

где каждое слагаемое  $V_i$  представляет собой функцию значения, ссылающуюся только на атрибут  $x_i$ . Например, вполне допустима такая возможность, что решение по размещению аэропорта может быть принято на основе следующей функции значения:

$$V(noise, cost, deaths) = -noise \times 10^4 - cost - deaths \times 10^{12}$$

Функция значения такого типа называется **аддитивной функцией значения**. Аддитивные функции представляют собой исключительно естественный способ описания любой функции значения агента и действительно правильно описывают многие реальные ситуации. И даже если свойство MPI, строго говоря, не соблюдается, что может иметь место при крайних значениях атрибутов, аддитивная функция значения все еще может предоставлять хорошую аппроксимацию для предпочтений агента. Такое утверждение особенно полно оправдывается, когда нарушения свойства MPI возникают в тех частях диапазонов атрибутов, которые редко встречаются на практике.

### **Предпочтения с неопределенностью**

Если в рассматриваемой проблемной области присутствует неопределенность, то необходимо также рассмотреть структуру предпочтений между лотереями и понять результирующие свойства функций полезности, а не просто функций значения. Математические основы решения этой проблемы могут оказаться весьма сложными, поэтому здесь мы представим только один из основных результатов, чтобы дать понять, как может быть решена эта проблема. Для ознакомления с исчерпывающим обзором работ в этой области рекомендуем читателю обратиться к [788].

Основное понятие **независимости полезностей** позволяет расширить понятие независимости предпочтений так, чтобы оно охватывало лотереи: множество атрибутов **X** является независимым по полезности от множества атрибутов **Y**, если предпочтения между лотереями по атрибутам **X** независимы от конкретных значений атрибутов **Y**. Множество атрибутов является **взаимно независимым по полезностям** (Mutually Utility-Independent — MUI), если каждое из его подмножеств является независимым по полезностям от остальных атрибутов. Опять-таки предположение о том, что атрибуты задачи с аэропортом обладают свойством MUI, кажется вполне резонным.

Из свойства MUI следует, что поведение агента может быть описано с помощью **мультипликативной функции полезности** [787]. С общей формой мультипликативной функции полезности можно проще всего ознакомиться, рассмотрев случай с тремя атрибутами. В целях сокращения мы будем использовать запись  $U_i$  для обозначения  $U_i(x_i)$ :

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_2 k_3 U_2 U_3 + k_3 k_1 U_3 U_1 + k_1 k_2 k_3 U_1 U_2 U_3$$

Хотя это соотношение на первый взгляд не кажется очень простым, оно содержит лишь три одноатрибутных функции полезности и три константы. Вообще говоря, любую  $n$ -атрибутную задачу, характеризующуюся наличием свойства MUI, можно промоделировать с использованием  $n$  одноатрибутных полезностей и  $n$  констант. Каждая из одноатрибутных функций полезности может быть разработана независимо от других атрибутов, а применение комбинации этих функций гарантирует формирование правильных общих предпочтений. Для получения чисто аддитивной функции полезности необходимо ввести некоторые дополнительные предположения.

## 16.5. СЕТИ ПРИНЯТИЯ РЕШЕНИЙ

В этом разделе рассматривается общий механизм принятия рациональных решений. Описанную здесь систему обозначений часто называют **диаграммами влияния** [695], но мы будем использовать более описательный термин **сети принятия решений**. В сетях принятия решений байесовские сети комбинируются с узлами дополнительных типов, которые обозначают действия и полезности. В качестве примера будет рассматриваться задача выбора площадки для строительства аэропорта.

### Способы представления задачи принятия решений с помощью сетей принятия решений

В своей наиболее общей форме любая сеть принятия решений представляет информацию о текущем состоянии агента, его возможных действиях, о состоянии, которое станет результатом данного действия агента, и о полезности этого состояния. Таким образом, данная сеть может служить основой для реализации агентов, действующих с учетом полезности, такого типа, который был впервые представлен в разделе 2.4. На рис. 16.5 показана сеть принятия решений в задаче выбора площадки для строительства аэропорта. Этот рисунок может служить иллюстрацией того, как используются узлы трех описанных ниже типов.

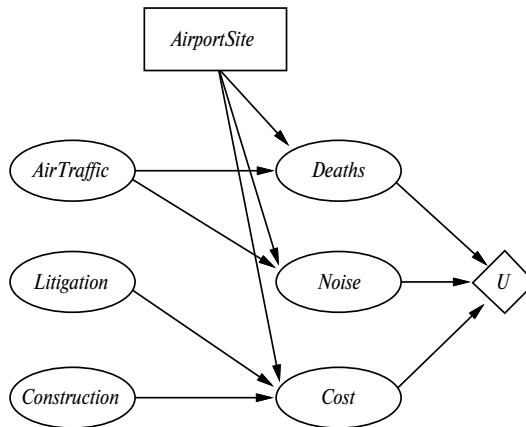


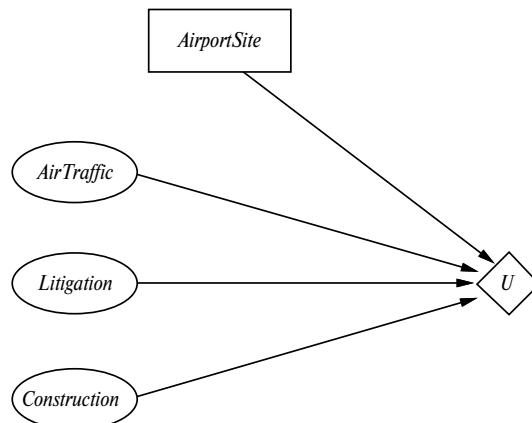
Рис. 16.5. Простая сеть принятия решений в задаче выбора площадки для строительства аэропорта

- ❖ **Узлы жеребьевки** (овалы) представляют собой случайные переменные, как в байесовских сетях. Агент может не иметь определенной информации о стоимости строительства, интенсивности воздушного трафика и о потенциальных возможностях урегулирования формальностей, связанных с получением разрешения на строительство, а также о значениях переменных *Deaths*, *Noise* и суммарной стоимости *Cost*, поскольку каждое из этих значений зависит от особенностей выбранной площадки. Каждый узел жеребьевки имеет связанное с ним распределение условных вероятностей, которое проиндексировано по состояниям его родительских узлов. В сетях принятия решений родительские узлы могут включать узлы принятия решений, а также узлы жеребьевки. Обратите внимание на то, что каждый из узлов жеребьевки в текущем состоянии может войти в состав более крупной байесовской сети, применяемой для оценки затрат на строительство, интенсивностей воздушного трафика или потенциальных возможностей формального урегулирования.
- ❖ **Узлы принятия решений** (прямоугольники) представляют собой точки, в которых лицу, принимающему решение, предоставляется выбор вариантов действий. В этом случае действие *AirportSite* может принимать различное значение для каждой площадки, подлежащей рассмотрению. Этот выбор влияет на стоимость, безопасность и шум, т.е. на те параметры, которые станут следствием строительства аэропорта. В данной главе предполагается, что нам придется иметь дело только с единственным узлом принятия решений, а в главе 17 рассматриваются случаи, в которых необходимо принимать больше одного решения.
- ❖ **Узлы полезности** (ромбы) представляют функцию полезности агента<sup>4</sup>. Родительскими переменными узла полезности являются все переменные, описывающие результат, который непосредственно влияет на полезность. С узлом

<sup>4</sup> Такие узлы в литературе часто называют **узлами значения**, но авторы предпочитают подчеркивать различие между функциями полезности и функциями значения, как было описано выше, поскольку результирующее состояние может представлять собой лотерею.

полезности связано описание полезности агента как функции от родительских атрибутов. Это описание может представлять собой табуляцию функции или может быть выражено в виде параметризованной аддитивной или мультилинейной функции.

Кроме того, во многих случаях применяется также упрощенная форма. Используемая при этом система обозначений остается неизменной, но исключаются узлы жеребьевки, описывающие результирующее состояние. Вместо этого узел полезности связывается непосредственно с узлами текущего состояния и с узлом принятия решений. В данном случае вместо представления функции полезности от состояний узел полезности представляет ожидаемую полезность, связанную с каждым действием, как было определено в уравнении 16.1. Поэтому такие таблицы авторы настоящей книги называют **таблицами “действие–полезность”**. На рис. 16.6 показано представление задачи с аэропортом в форме “действие–полезность”.



*Рис. 16.6. Упрощенное представление задачи выбора площадки для строительства аэропорта. Исключены узлы жеребьевки, соответствующие результирующим состояниям*

Обратите внимание на то, что узлы жеребьевки *Noise*, *Deaths* и *Cost*, показанные на рис. 16.5, ссылаются на будущее состояние, поэтому их значения ни в коем случае не должны определяться в виде переменных свидетельства. Таким образом, упрощенную версию, в которой исключены эти узлы, можно использовать во всех тех случаях, когда допустимо использование более общей формы. Однако, несмотря на то, что в упрощенной форме содержится меньше узлов, исключение явного описания результатов решения по выбору площадки означает, что такая сеть является менее гибкой по отношению к возможным изменениям обстоятельств. Например, на рис. 16.5 изменение допустимых уровней шума самолета можно отразить в виде изменения в таблице условных вероятностей, связанной с узлом *Noise*, тогда как изменение веса, касающегося компонента с описанием шумового загрязнения в функции полезности, может быть отражено с помощью изменения в таблице полезности. С другой стороны, в схеме “действие–полезность”, приведенной на рис. 16.6, все такие изменения должны быть отражены в виде изменений в таблице

“действие–полезность”. По сути формулировка на основе “действия–полезности” представляет собой откомпилированную версию первоначальной формулировки.

### **Вычисления с помощью сетей принятия решений**

Действия выбираются путем проведения с помощью сети принятия решений соответствующих вычислений для каждого возможного ряда значений узла принятия решений. После того как определено значение узла принятия решений, он ведет себя полностью аналогично узлу жеребьевки, которому были присвоены значения по такому же принципу, как переменной свидетельства. Алгоритм проведения вычислений в сетях принятия решений описан ниже.

1. Определить значения переменных свидетельства для текущего состояния.
2. Для каждого возможного значения узла принятия решений:
  - а) ввести это значение в узел принятия решений;
  - б) вычислить апостериорные вероятности для родительских узлов узла полезности, используя стандартный алгоритм вероятностного вывода;
  - в) вычислить результирующее значение полезности для данного действия.
3. Возвратить действие с самым высоким значением полезности.

Этот алгоритм представляет собой непосредственное расширение алгоритма вычислений в байесовской сети и может быть внедрен непосредственно в проект агента, приведенный в листинге 13.1. Как будет показано в главе 17, эта задача становится намного более интересной, когда существует возможность последовательного выполнения нескольких действий.

## **16.6. СТОИМОСТЬ ИНФОРМАЦИИ**

В приведенном выше анализе предполагалось, что агенту, прежде чем он приступает к принятию решения, предоставляется вся относящаяся к делу информация или, по меньшей мере, вся доступная информация. Но на практике такая ситуация возникает чрезвычайно редко.  Одной из наиболее важных составляющих процесса принятия решений являются знания о том, какие вопросы следует задавать. Например, врач не может рассчитывать на то, что ему будут предоставлены результаты всех возможных диагностических тестов и опросов к тому времени, как пациент впервые войдет в его кабинет<sup>5</sup>. Медицинские тесты часто являются дорогостоящими, а иногда даже опасными (такая опасность может возникать, во-первых, непосредственно, а во-вторых, из-за связанных с ними задержек). Важность проведения медицинских тестов зависит от двух факторов: от того, в какой степени получение результатов этих тестов приведут к разработке значительно лучшего плана решения, а также от того, насколько велика вероятность различных результатов тестов.

В настоящем разделе описана  теория стоимости информации, которая позволяет любому агенту решить, какую информацию он должен приобрести. Получение

---

<sup>5</sup> В США единственным вопросом, который всегда задают заранее, является то, есть ли у пациента страховка.

информации осуществляется с помощью **действий по восприятию** (см. главу 12). Поскольку функция полезности агента редко ссылается на содержимое внутреннего состояния агента, притом что общее назначение действий по восприятию сводится к оказанию влияния на это внутреннее состояние, мы должны оценивать качество действий по восприятию на основании того, какое влияние они оказывают на последующие “реальные” действия агента. Таким образом, теория стоимости информации связана с определенной формой последовательного принятия решений.

### Простой пример

Предположим, что нефтедобывающая компания надеется приобрести лицензию на один из  $n$  равнозначных по своей перспективности участков океанского шельфа для проведения разведочных работ. Кроме того, примем дополнительное предположение, что точно один и только один из этих участков обладает запасами нефти стоимостью  $C$  долларов и что цена каждого участка равна  $C/n$  долларов. Если эта компания нейтрально относится к риску, то она должна быть безразлична к выбору между покупкой лицензии на один из участков и отказом от такой покупки.

А теперь допустим, что некий сейсмолог доложил руководству этой компании результаты проведенного исследования участка номер 3, которые определенно показывают, что на этом участке имеется нефть. Какую сумму должна быть готова компания заплатить за эту информацию? Один из способов получения ответа на этот вопрос состоит в том, чтобы проверить, какие действия предприняла бы компания, обладая указанной информацией, как описано ниже.

- С вероятностью  $1/n$  исследование покажет наличие нефти на участке 3. В этом случае компания купит участок 3 за  $C/n$  долларов и получит прибыль в  $C - C/n = (n-1)C/n$  долларов.
- С вероятностью  $(n-1)/n$  исследование покажет, что участок не содержит нефти, и в этом случае компания должна купить другой участок. Теперь вероятность обнаружения нефти на одном из других участков измеряется значением от  $1/n$  до  $1/(n-1)$ , поэтому компания получит ожидаемую прибыль  $C/(n-1) - C/n = C/n(n-1)$  долларов.

Теперь мы можем рассчитать ожидаемую прибыль при наличии информации о результатах исследования:

$$\frac{1}{n} \times \frac{(n-1)C}{n} + \frac{n-1}{n} \times \frac{C}{n(n-1)} = C/n$$

Поэтому компания должна быть готова заплатить сейсмологу за эту информацию вплоть до  $C/n$  долларов, поскольку данная информация стоит столько же, сколько и сам участок.

Стоимость информации определяется тем фактом, что при наличии такой информации можно изменить собственную стратегию таким образом, чтобы она соответствовала действительной ситуации. Наличие информации позволяет выявить отличительные особенности рассматриваемой ситуации, а без этой информации в лучшем случае можно лишь найти среднее значение по всем возможным ситуациям. Вообще говоря, стоимость данного конкретного фрагмента информации опре-

деляется той разницей в ожидаемых значениях между наилучшими действиями, которая возникает до и после получения этой информации.

## Общая формула

Общую математическую формулу для стоимости информации получить несложно. Обычно предполагается, что информация — это полученное точное свидетельство, касающееся значения некоторой случайной переменной  $E_j$ , поэтому используется выражение **стоимость полной информации** (Value of Perfect Information — VPI)<sup>6</sup>. Обозначим текущие знания агента как  $E$ . В таком случае стоимость текущего наилучшего действия  $\alpha$  определяется с помощью соотношения

$$EU(\alpha | E) = \max_A \sum_i U(Result_i(A)) P(Result_i(A) | Do(A), E)$$

а стоимость нового наилучшего действия (после получения нового свидетельства  $E_j$ ) принимает вид

$$EU(\alpha_{E_j} | E, E_j) = \max_A \sum_i U(Result_i(A)) P(Result_i(A) | Do(A), E, E_j)$$

Но  $E_j$  — это случайная переменная, значение которой в настоящее время неизвестно, поэтому необходимо выполнить усреднение по всем возможным значениям  $e_{jk}$  переменной  $E_j$ , которые только могут быть обнаружены, с использованием текущих степеней уверенности в том, что эта переменная имеет данное значение. Поэтому стоимость обнаружения значения  $E_j$  при наличии текущей информации  $E$  определяется следующим образом:

$$VPI_E(E_j) = \left( \sum_k P(E_j=e_{jk} | E) EU(\alpha_{e_{jk}} | E, E_j=e_{jk}) \right) - EU(\alpha | E)$$

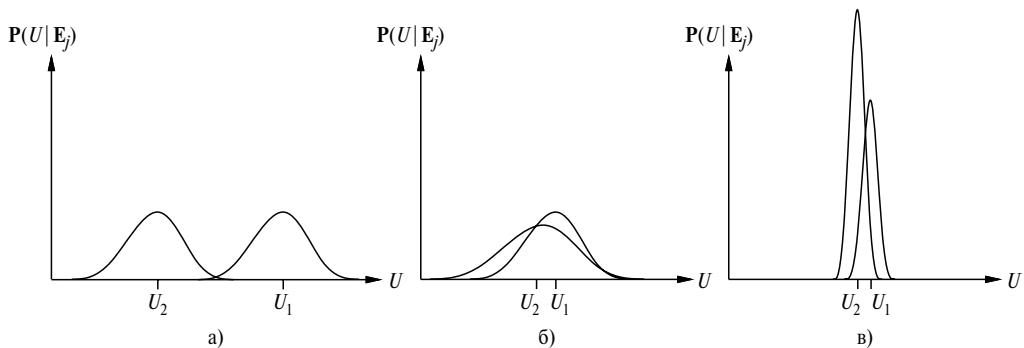
Для того чтобы сформировать определенное интуитивное представление о том, какой смысл имеет эта формула, рассмотрим простой случай, в котором имеются только два действия,  $A_1$  и  $A_2$ , подлежащих выбору в процессе принятия решений. Текущими ожидаемыми полезностями этих действий являются  $U_1$  и  $U_2$ . Получение информации  $E_j$  приведет к определению каких-то новых значений ожидаемых полезностей  $U_1'$  и  $U_2'$  для этих действий, но прежде чем будет получена информация  $E_j$ , уже будут существовать определенные распределения вероятностей по всем возможным значениям  $U_1'$  и  $U_2'$  (которые, согласно принятому предположению, являются независимыми).

Предположим, что действия  $A_1$  и  $A_2$  представляют собой прохождение по двум разным маршрутам через горную цепь зимой. В действии  $A_1$  предусмотрен проезд по удобному прямому пути через невысокий перевал, а в действии  $A_2$  — проезд по извилистой грунтовой дороге через вершину. Даже если дана только эта информация,

---

<sup>6</sup> Неполную информацию о переменной  $X$  можно промоделировать с помощью полной информации о переменной  $Y$ , которая имеет вероятностную связь с переменной  $X$ ; пример применения такого подхода приведен в упр. 16.11.

очевидно, что предпочтительным является действие  $A_1$ , поскольку весьма вероятно, что второй маршрут может быть заблокирован из-за лавин, и вместе с тем весьма маловероятно, что первый маршрут заблокирован из-за интенсивного трафика. Поэтому очевидно, что полезность  $U_1$  выше по сравнению с  $U_2$ . Но возможно, что результаты наблюдений со спутника  $E_j$ , касающиеся фактического состояния каждой дороги, приведут к получению новых оценок двух путей через горный хребет,  $U_1'$  и  $U_2'$ . Распределения вероятностей для исходных ожиданий показаны на рис. 16.7, а. Очевидно, что в этом случае нет смысла оплачивать получение данных со спутника, поскольку маловероятно, что полученная с их помощью информация приведет к изменению плана. Если получение новой информации не приводит к каким-либо изменениям, то она не имеет реальной стоимости.



*Рис. 16.7. Три наиболее общих случая, касающихся определения стоимости информации: действие  $A_1$  почти со всей определенностью будет оставаться лучшим по сравнению с действием  $A_2$ , поэтому дополнительная информация не нужна (а); выбор неясен, поэтому информация крайне важна (б); выбор неясен, но поскольку разница между полезностями действий невелика, информация не имеет такой уж большой ценности (в)*

А теперь предположим, что необходимо выбрать одну из двух извилистых грунтовых дорог, имеющих немного разную длину, для того чтобы вывезти за перевал пациента с серьезной травмой. В таком случае, даже если полезности  $U_1$  и  $U_2$  достаточно близки, распределения вероятностей  $U_1'$  и  $U_2'$  являются очень широкими. Существует значительная вероятность того, что второй маршрут окажется свободным, а первый — заблокированным, и в этом случае разность между полезностями будет очень большой. Формула VPI показывает, что в данном случае вполне оправдано получение отчета со спутника. Подобная ситуация показана на рис. 16.7, б).

Далее, предположим, что осуществляется выбор между двумя грунтовыми дорогами летом, когда их блокировка из-за лавин является маловероятной. В этом случае отчеты со спутника могут показать, что одна дорога является более живописной, чем другая, из-за того, что вдоль нее расположены цветущие альпийские луга, или, возможно, немного более влажной из-за блуждающих потоков. Поэтому весьма вероятно, что турист, путешествующий через горы, изменил бы свой план, если бы имел подобную информацию. Но и в таком случае разница в значениях между двумя маршрутами все еще, по-видимому, остается очень небольшой, поэтому нет смысла тратить лишние усилия на получение отчетов со спутника. Такая ситуация показана на рис. 16.7, в).

Подводя итог, можно сказать, что  $\Leftrightarrow$  стоимость информации определяется той степенью, в которой она может вызвать изменение плана, а также той степенью, в какой новый план окажется значительно лучше по сравнению со старым.

### Свойства показателей стоимости информации

Напрашивается очевидный вопрос — возможно ли, чтобы информация была вредной, иными словами, может ли она фактически иметь отрицательную ожидаемую стоимость? Интуитивно ясно, что такая ситуация невозможна. В конечном итоге в наихудшем случае можно просто проигнорировать ненужную информацию и сделать вид, как будто она так и не была получена. Данное интуитивное представление подтверждается приведенной ниже теоремой, которая относится к любому агенту, действующему на основе теории принятия решений.

$\Leftrightarrow$  Стоимость информации является неотрицательной:

$$\forall j, E \quad VPI_E(E_j) \geq 0$$

Эта теорема следует непосредственно из определения свойства VPI, и мы оставляем доказательство этой теоремы читателю в качестве упражнения (упр. 16.12). Важно помнить, что свойство VPI зависит от текущего состояния информации, именно поэтому в формулировке данной теоремы это свойство обозначено подстрочным индексом. Значение VPI по мере получения дополнительной информации может изменяться. В крайнем случае оно может стать равным нулю, если рассматриваемая переменная уже имеет известное значение. Таким образом, свойство VPI не аддитивно. Это означает, что справедливо следующее соотношение:

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E, E_j}(E_k) \quad (\text{в общем случае})$$

Но свойство VPI не зависит от порядка рассматриваемых в нем переменных, что должно быть интуитивно ясно. Это означает, что справедливо такое соотношение:

$$VPI_E(E_j, E_k) = VPI_E(E_j) + VPI_{E, E_j}(E_k) = VPI_E(E_k) + VPI_{E, E_k}(E_j)$$

В силу того, что действия по восприятию не зависят от последовательности их выполнения, они отличаются от обычных действий, а задача вычисления значения последовательности действий по восприятию упрощается.

### Реализация агента, действующего на основе сбора информации

Успешно действующий агент должен задавать вопросы пользователю в приемлемом порядке, не требовать ответов на вопросы, не относящиеся к делу, учитывать важность каждого фрагмента информации применительно к стоимости его получения и прекращать задавать вопросы, когда сложившаяся обстановка требует перехода к другим действиям. Все эти возможности можно воплотить в проекте агента, используя в качестве критерия стоимость информации.

В листинге 16.1 показан общий проект агента, который способен осуществлять интеллектуальный сбор информации, прежде чем приступить к действиям. На данный момент мы будем предполагать, что с каждой наблюдаемой переменной свидетельства  $E_j$  связана соответствующая стоимость  $Cost(E_j)$ , которая отражает стоимость получения этого свидетельства с помощью проведения тестов, организации консультаций, получения ответов на дополнительные вопросы или других подобных

действий. Агент запрашивает фрагменты информации, которые представляются для него наиболее ценными по сравнению с их стоимостью. Предполагается, что результатом действия по осуществлению запроса  $\text{Request}(E_j)$  является то, что следующий результат восприятия предоставляет значение  $E_j$ . А если ни одно наблюдение не оправдывает его стоимость, агент выбирает “реальное” действие.

**Листинг 16.1.** Проект простого агента, действующего на основе сбора информации. Этот агент функционирует, снова и снова выбирая наблюдение с наивысшим информационным значением, до тех пор, пока стоимость следующего наблюдения не станет выше по сравнению с ожидаемой от него пользой

---

```
function Information-Gathering-Agent(percept) returns действие action
  static: D, сеть принятия решений

  включить данные о восприятии percept в сеть D
  j  $\leftarrow$  значение, максимизирующее выражение  $VPI(E_j) - Cost(E_j)$ 
  if  $VPI(E_j) > Cost(E_j)$ 
    then return  $\text{Request}(E_j)$ 
  else return наилучшее действие из D
```

---

Описанный здесь алгоритм агента реализует один из подходов к сбору информации, называемый **близоруким**. Это связано с тем, что в данном подходе формула VPI используется без дальновидных расчетов и значение информации определяется так, как будто было бы достаточно получить значение единственной переменной свидетельства. А если нет ни одной переменной свидетельства, которая оказала бы значительную помощь, близорукий агент может преждевременно приступить к действиям, тогда как было бы лучше вначале запросить значения еще двух или нескольких переменных и только после этого начинать действовать. Близорукие методы управления основаны на той же эвристической идее, что и жадный поиск, и часто хорошо работают на практике (например, было показано, что подобные системы управления превосходят по своей производительности опытных врачей, когда речь идет о подборе необходимых диагностических тестов). Но агент, действующий на основе сбора информации, который является идеально рациональным, должен рассматривать все возможные последовательности информационных запросов, приводящие к внешнему действию, а также все возможные результаты этих запросов. Поскольку содержание второго запроса зависит от результатов первого запроса, агент должен исследовать пространство **условных планов**, как было описано в главе 12.

## 16.7. ЭКСПЕРТНЫЕ СИСТЕМЫ, ОСНОВАННЫЕ НА ИСПОЛЬЗОВАНИИ ТЕОРИИ ПРИНЯТИЯ РЕШЕНИЙ

В научной области **анализа проблемы принятия решений**, которая бурно развивалась в 1950–1960-х годах, изучалось применение теории принятия решений к реальным задачам принятия решений. Результаты, полученные в этой сфере, использовались для упрощения задачи принятия рациональных решений во многих важных проблемных областях, где ставки достаточно высоки, таких как бизнес, государственное управление, юриспруденция, военная стратегия, медицинская ди-

агностика и здравоохранение, техническое проектирование и управление ресурсами. Применяемый при этом процесс предусматривал тщательное исследование всех действий и результатов, а также учет предпочтений, назначенных каждому результату. По традиции в области анализа проблем принятия решений специалисты подразделяются на две категории: **лицо, принимающее решения**, формулирует отношения предпочтений между результатами, а **лицо, анализирующее решения**, составляет список возможных действий и результатов, а также получает от лица, принимающего решения, информацию о предпочтениях, чтобы определить наилучшую стратегию. До начала 1980-х годов основное назначение анализа решений состояло в том, чтобы помочь людям принимать решения, которые действительно отражают их собственные предпочтения. А в настоящее время, когда процессы принятия решений во все большей степени становятся автоматизированными, анализ решений используется для обеспечения того, чтобы эти автоматизированные процессы действовали в соответствии с предъявленными к ним требованиями.

Как было описано в главе 14, на первых порах исследования в области экспертных систем сосредоточивались на получении ответов на вопросы, а не на принятии решений. Такие системы, которые рекомендуют действия, а не предоставляют оценки сложившихся обстоятельств, как правило, функционируют на основе правил “условие–действие”, а не с использованием явных представлений результатов и предпочтений. Но после разработки теории байесовских сетей в конце 1980-х годов появилась возможность создавать крупномасштабные системы, способные формировать обоснованные вероятностные выводы на основании полученных свидетельств. А внедрение в практику сетей принятия решений означало появление возможности создания экспертных систем, способных рекомендовать оптимальные решения с учетом предпочтений пользователя, а также доступных свидетельств.

Система, в которой учитываются полезности, способна избежать одной из наиболее распространенных ловушек, связанных с процессом предоставления консультаций, — смешивания правдоподобия и важности. Например, наиболее широко применяемой стратегией в ранних медицинских экспертных системах было ранжирование возможных диагнозов в порядке правдоподобия и выдача сообщения о наиболее вероятном диагнозе. К сожалению, такой подход иногда оказывался катастрофическим! Для большинства пациентов в общей медицинской практике двумя наиболее вероятными диагнозами обычно являются следующие: “У вас нет ничего серьезного” и “Вы сильно простудились”, но если третьим по степени вероятности диагнозом для данного конкретного пациента (который тем не менее не был сообщен системой) является рак легких — это уже серьезная ситуация! Очевидно, что план медицинского обследования или лечения должен зависеть и от учета вероятностей, и от учета полезностей.

Теперь мы опишем процесс инженерии знаний для экспертных систем, основанных на теории принятия решений. В качестве примера рассмотрим задачу выбора медицинского лечения для определенного вида сердечного заболевания у детей обоих полов (см. работу Лукаса [962]).

Около 0,8% детей рождаются с той или иной сердечной аномалией, причем наиболее распространенной из них является **коарктация аорты** (сужение аорты). Этот дефект можно исправить с помощью хирургического вмешательства, ангиопластики (расширения аорты с помощью надувного баллона, помещенного внутри этой артерии) или терапевтического лечения. Проблема состоит в том, чтобы определить, ка-

кой способ лечения следует использовать и когда его проводить: чем младше ребенок, тем выше риск, связанный с применением определенных способов лечения, но ожидание также не должно продолжаться слишком долго. Экспертная система, основанная на использовании теории принятия решений, которая предназначена для решения данной проблемы, может быть создана группой специалистов, состоящей по меньшей мере из одного специалиста в данной проблемной области (детского кардиолога) и одного инженера по знаниям. Сам этот процесс можно разбить на следующие этапы (который читатель может сравнить с этапами разработки системы на основе логики, описанными в разделе 8.4).

- **Создание причинной модели.** Определить, каковыми являются возможные симптомы, нарушения, способы лечения и результаты. Затем провести между ними дуги, указывающие, какими нарушениями вызваны те или иные симптомы и какие способы лечения позволяют устраниить те или иные нарушения. Некоторые из этих сведений должны быть хорошо известны специалисту в данной проблемной области, а другие сведения он может почерпнуть из литературы. Эта модель часто во многом напоминает неформальные графические описания, приведенные в медицинских учебниках.
- **Упрощение причинной модели до уровня качественной модели принятия решений.** Поскольку применяемая модель предназначена для принятия решений, касающихся способа лечения, а не для других целей (таких как определение совместной вероятности некоторых комбинаций симптом/нарушение), часто можно упростить причинную модель, удаляя переменные, которые не требуются в процессе принятия решений, касающихся лечения. Кроме того, иногда возникает необходимость разделить или соединить переменные, чтобы они соответствовали интуитивным представлениям специалиста. Например, предположим, что первоначальная модель коарктации аорты включала переменную *Treatment* (Лечение) со значениями *surgery* (хирургическое вмешательство), *angioplasty* (ангиопластика) и *medication* (терапевтическое лечение), а также отдельную переменную *Timing* для определения сроков выполнения процедуры лечения. Но допустим, что специалисту нелегко рассматривать все эти переменные отдельно, поэтому они были объединены так, что переменная *Treatment* принимает значения типа *surgery in 1 month* (проведение хирургического вмешательства в течение одного месяца). В результате будет получена модель, приведенная на рис. 16.8.
- **Присвоить значения вероятностей.** Вероятности можно определить по базам данных о пациентах, по результатам исследований, приведенным в литературе, или на основании субъективных оценок самого специалиста. В тех случаях, если в литературе даны сведения о вероятностях не совсем подходящих типов, для вычисления требуемых вероятностей можно применить такие методы, как правило Байеса и маргинализация. Исследования показали, что специалисты в проблемной области в большей степени обладают способностью оценить вероятность результата, если дана причина (например,  $P(\text{dyspnoea} | \text{heart failure})$ ), а не наоборот.
- **Присвоение значений полезностей.** Если количество возможных результатов невелико, то их можно перечислить и оценить отдельно. Допустим, что в дан-

ном случае создается шкала результатов от наилучшего к наихудшему и каждому из них присваивается числовое значение, например случаю смерти присваивается значение  $-1000$ , а полному излечению — значение  $0$ . После этого на данную шкалу можно поместить другие результаты. Такая задача может быть выполнена специалистом в данной проблемной области, но лучше, если в ее решении примет участие пациент (или, если речь идет о детях, то родители пациентов), поскольку разные люди имеют различные предпочтения. Если же количество результатов растет экспоненциально, то может потребоваться определенный способ, позволяющий комбинировать их с использованием многоатрибутных функций полезности. Например, можно утверждать, что отрицательная полезность различных осложнений является аддитивной.

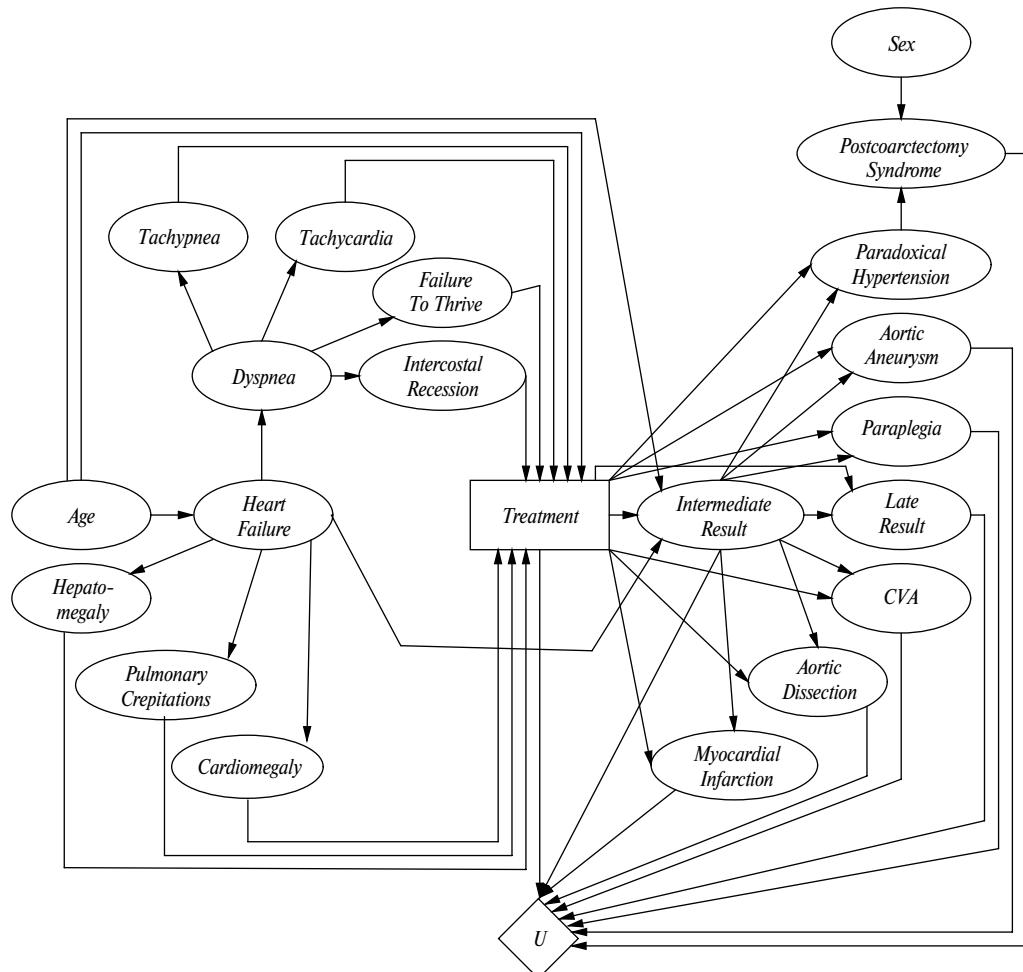


Рис. 16.8. Диаграмма влияний для задачи принятия решений, касающихся коарктации аорты (предоставлена Питером Лукасом)

- **Проверка и уточнение модели.** Для проверки работоспособности системы потребуется множество правильных пар “вход–выход” — (*input*, *output*) — так называемый ~~и~~ золотой стандарт, с которым можно выполнить сравнение. Для медицинских экспертных систем это обычно означает, что можно собрать самых лучших врачей, дать им на рассмотрение несколько случаев и попросить у них сообщить их диагноз и рекомендуемый план лечения. Затем осуществляется проверка того, насколько результаты, полученные данной системой, согласуются с указанными рекомендациями. Если согласование оказывается неприемлемым, то нужно попытаться выделить ее части, которые действуют неправильно, и провести их доработку. Может также оказаться полезным выполнение прогона системы “в обратном направлении”: вместо того чтобы вводить в систему симптомы и запрашивать диагнозы, можно представить ей диагноз, такой как “сердечная недостаточность”, определить предсказанную вероятность симптомов, таких как тахикардия, и сравнить это значение с данными, полученными из медицинской литературы.
- **Осуществление анализа чувствительности.** На этом важном этапе проверяется, чувствительно ли наилучшее решение к небольшим изменениям в присвоенных значениях вероятностей и полезностей; для этого систематически осуществляется варьирование этих параметров и повторное проведение вычислений. Если небольшие изменения приводят к получению намного отличающихся решений, то может потребоваться затратить больше ресурсов на сбор более качественных данных. Если же все небольшие изменения приводят к получению одного и того же решения, то пользователь будет больше доверять этому решению как правильному. Анализ чувствительности особенно важен, поскольку одно из основных критических замечаний в адрес вероятностных подходов к созданию экспертных систем состоит в том, что трудно оценить необходимые числовые значения вероятностей. Анализ чувствительности часто показывает, что достаточно задать многие числовые данные только очень приблизительно. Например, мы можем не знать, каково значение априорной вероятности  $P(\text{tachycardia})$ , но после опробования многих разных значений этой вероятности и получения в каждом случае одинакового рекомендованного действия по диаграмме влияния можно меньше беспокоиться о том, что отсутствует соответствующая достоверная информация.

## 16.8. РЕЗЮМЕ

В данной главе показано, как объединить теорию полезности с теорией вероятностей, чтобы дать возможность агенту выбрать действия, которые максимизируют его ожидаемую производительность.

- **Теория вероятностей** описывает, в чем должен быть уверен агент согласно полученному свидетельству, **теория полезности** показывает, к чему должен стремиться агент, а **теория принятия решений** позволяет объединить подходы этих двух теорий для определения того, что должен делать агент.
- Теория принятия решений может использоваться для создания систем, которые принимают решения, рассматривая все возможные действия и выбирая из

них именно то, которое приводит к наилучшему ожидаемому результату. Такая система известна под названием **рационального агента**.

- Теория полезностей показывает, что агент, руководствующийся отношениями предпочтения между лотереями, совместимыми с множеством простых аксиом, может быть описан как обладающий функцией полезности; кроме того, агент выбирает действия так, чтобы можно было максимизировать его ожидаемую полезность.
- **Теория многоатрибутной полезности** посвящена изучению полезности, которая зависит от нескольких разных атрибутов состояний. **Стохастическое доминирование** представляет собой особенно удобный метод принятия непротиворечивых решений даже при отсутствии точных значений полезности для атрибутов.
- **Сети принятия решений** представляют собой простую формальную систему для описания и решения задач принятия решений. Они являются естественным расширением байесовских сетей и, кроме узлов жеребьевки, содержат узлы решения и узлы полезности.
- Иногда для решения задачи приходится заниматься поиском дополнительной информации, прежде чем принимать решение. **Стоймость информации** определена как ожидаемое повышение полезности по сравнению с принятием решений без этой информации.
- **Экспертные системы**, в которых предусматривается использование информации о полезности, обладают дополнительными возможностями по сравнению с системами, в которых применяется исключительно вероятностный вывод. Они не только обладают способностью вырабатывать решения, но и могут использовать стоимость информации для определения того, следует ли стремиться к ее получению, а также способны рассчитать чувствительность своих решений к небольшим изменениям в оценках вероятности и полезности.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Одним из первых примеров применения принципа максимальной ожидаемой полезности (хотя и не совсем правильным, поскольку в нем использовались бесконечные значения полезности) был рассказ о пари Паскаля, впервые опубликованный в книге *Port-Royal Logic* [40]. Даниил Бернулли [111], исследовавший санкт-петербургский парадокс (см. упр. 16.3), был первым, кто понял важность измерения предпочтений в отношении лотерей и написал такие слова: “*Стоймость* любого предмета должна быть основана не на его *цене*, а на той *пользе*, которую он может принести” (курсив Бернулли). Джереми Бентам [100] предложил **гедонистическое исчисление** для взвешивания “удовольствий” и “неприятностей”, доказывая, что все решения (а не только касающиеся денег) можно свести к сравнению полезностей.

Определение числовых значений полезности на основе предпочтений было впервые выполнено Рамзеем [1265]; аксиомы предпочтений, приведенные в настоящей книге, ближе всего по своей форме тем, которые были вновь открыты в книге *Theory of Games and Economic Behavior* [1546]. Хорошее описание этих аксиом в рамках дискуссии о предпочтении риска приведено в [693]. Рамзей предложил способ вычисле-

ния субъективных вероятностей (а не только полезностей) на основе предпочтений агента; Сэвеж [1354] и Джейфри [727] предложили более современные вычислительные конструкции такого рода. В [1547] изложены современные перспективы анализа проблемы принятия решений и показана ее связь со структурами предпочтений людей. Такая мера полезности, как микрошанс смерти, обсуждается в [694]. В обзоре, опубликованном журналом *Economist* в 1994 году, указано, что стоимость жизни находится в пределах от 750 000 долларов и до 2,6 миллиона долларов. Однако Ричард Талер [1503] обнаружил нерациональные отклонения в оценках той денежной суммы, которую люди готовы заплатить, чтобы избежать риска смерти, в сравнении с суммой, которую они готовы получить, чтобы принять на себя этот риск. С вероятностью 1/1000 его респонденты не заплатили бы больше 200 долларов, чтобы исключить риск, но не были согласны получить даже 50 000 долларов, чтобы подвергнуться риску.

В принятии решений, касающихся политики в области здравоохранения и социального обеспечения, гораздо более широкое распространение находит такой показатель, как QALY, а не микрошанс смерти; типичный пример обоснования одного из крупных изменений политики в области общественного здравоохранения на основе повышения ожидаемой полезности, измеряемого в единицах QALY, приведен в [1320].

Книга *Decisions with Multiple Objectives: Preferences and Value Tradeoffs* [788] содержит исчерпывающее введение в теорию многоатрибутной полезности (MultiAttribute Utility Theory — MAUT). В ней описаны первые компьютерные реализации методов выявления необходимых параметров для многоатрибутной функции полезности и приведены исчерпывающие отчеты о результатах практического применения этой теории. В области искусственного интеллекта основным источником сведений о теории многоатрибутной полезности является статья Уэллмана [1572], которая включает описание системы URP (Utility Reasoning Package), позволяющей использовать коллекцию высказываний о независимости предпочтений и условной независимости для анализа структуры задач принятия решений. Результаты обширного исследования в области использования понятия стохастического доминирования в сочетании с качественными вероятностными моделями приведены в [1573], [1574]. В [1578] изложен предварительный набросок подхода к созданию метода, с помощью которого можно использовать сложный набор отношений независимости полезностей для создания структурированной модели функции полезности, во многом аналогично тому, как байесовские сети позволяют создавать структурированные модели совместных распределений вероятностей. В [52], [53] и [876] приведены дальнейшие результаты, полученные в данном научном направлении.

Теория принятия решений является стандартным инструментальным средством в экономике, финансах и науке управления с 1950-х годов. До 1980-х годов основным средством, применяемым для представления простых задач принятия решений, были деревья решений. В [1438] приведен обзор методологии анализа проблемы принятия решений. Сети принятия решений, или диаграммы влияния, были разработаны Говардом и Матесоном [695] на основе одной из ранних работ, выполненной группой специалистов (включая Говарда и Матесона) в институте SRI [1051]. Метод Говарда и Матесона обеспечивает формирование дерева решений на основе сети принятия решений, но в общем случае сформированное дерево имеет экспоненциальные размеры. Шахтер [1383] разработал метод принятия решений, основанный исключительно на использовании сети принятия решений, без создания промежу-

точного дерева решений. Этот алгоритм оказался также одним из первых алгоритмов, который обеспечивает полный вероятностный вывод в многосвязных байесовских сетях. В недавно опубликованной работе Нильссона и Лауритцена [1139] показано, какое отношение имеют алгоритмы для сетей принятия решений к продолжающимся разработкам в области создания алгоритмов кластеризации для байесовских сетей. В сборнике статей [1155] приведен целый ряд полезных статей по сетям принятия решений, как и в специальном выпуске журнала *Networks*, который вышел в 1990 году. Кроме того, статьи по сетям принятия решений и моделированию полезностей регулярно публикуются в журнале *Management Science*.

Теория стоимости информации была впервые проанализирована Роном Говардом [692]. Его статья оканчивается замечанием: “Если теория стоимости информации и связанные с ней структуры теории принятия решений не составят в будущем значительную часть образования инженеров, то люди с инженерными профессиями уступят свою традиционную роль в управлении научными и экономическими ресурсами в пользу людей, посвятивших себя другим профессиям”. До настоящего времени предсказанная им революция в области методов управления еще не наступила, но такое может случиться после того, как получат более широкое распространение методы использования теории стоимости информации в байесовских экспертных системах.

Как это ни удивительно, но лишь немногие исследователи в области искусственного интеллекта приняли на вооружение инструментальные средства теории принятия решений после появления самых первых приложений принятия решений в медицине, которые описаны в главе 13. Одним из немногих исключений стал Джерри Фельдман, который применил теорию принятия решений в задачах машинного зрения [461] и планирования [460]. После возрождения интереса к вероятностным методам в искусственном интеллекте в 1980-х годах получили широко распространение экспертные системы на основе теории принятия решений [686]. Фактически начиная с 1991 года и до настоящего времени на обложке журнала *Artificial Intelligence* изображается сеть принятия решений, хотя и создается впечатление, что автор рисунка этой сети позволил себе некоторые художественные вольности при выборе направления стрелок.

## УПРАЖНЕНИЯ

- 16.1.** (*Адаптировано из книги Дэвида Хекермана.*) Это упражнение касается содержания игры **Almanac Game** (игра на знание фактов из справочника), которая используется аналитиками решений для калибровки числовых оценок. На каждый из приведенных ниже вопросов дайте наилучший предполагаемый вами ответ, т.е. число, которое, по вашему мнению, с такой же вероятностью является слишком большим, с какой оно может быть слишком малым. Кроме того, приведите вашу гипотезу с оценкой на уровне 25-й процентили, т.е. такое число, которое, по вашему мнению, имеет 25% шансов на то, чтобы быть слишком высоким, и 75% шансов на то, чтобы быть слишком низким. Приведите такое же значение и для 75-й процентили. (Таким образом, вы должны дать всего три оценки для каждого вопроса — низкую, среднюю и высокую.)

- а) Количество пассажиров, которые совершили полеты между Нью-Йорком и Лос-Анджелесом в 1989 году.
- б) Население Варшавы в 1992 году.
- в) Год, в который испанский конкистадор Коронадо открыл реку Миссисипи.
- г) Количество голосов, полученных Джимми Картером во время президентских выборов в 1976 году.
- д) Возраст самого старого живого дерева по состоянию на 2002 год.
- е) Высота плотины Гувера (Hoover Dam) в футах.
- ж) Количество яиц, произведенных в штате Орегон в 1985 году.
- з) Количество буддистов в мире в 1992 году.
- и) Количество смертных случаев из-за СПИДа в Соединенных Штатах в 1981 году.
- к) Количество американских патентов, выданных в 1901 году.

Правильные ответы приведены после последнего упражнения этой главы. С точки зрения анализа решений интересно не то, насколько ваши средние предположения подошли к реальным ответам, но, скорее, то, насколько часто реальный ответ попал в установленные вами границы 25 и 75%. Если такая ситуация возникла примерно в половине случаев, то указанные вами границы были достаточно точными. Но если вы похожи на большинство людей, то проявите больше самоуверенности, чем следует, и более половины ответов выйдет за пределы этих границ. Постоянно практикуясь, вы сможете откалибровать свои оценки, чтобы устанавливаемые пределы стали более реалистичными и тем самым приносили больше пользы при предоставлении информации для принятия решений. Попробуйте ответить на второй ряд вопросов и определите, достигнуты ли вами какие-либо улучшения.

- а) Год рождения актрисы За За Гabor (Zsa Zsa Gabor).
  - б) Максимальное расстояние от Марса до Солнца в милях.
  - в) Стоимость в долларах пшеницы, экспортированной из Соединенных Штатов в 1992 году.
  - г) Количество тонн груза, обработанных в порту Гонолулу в 1991 году.
  - д) Ежегодный заработок в долларах губернатора Калифорнии в 1993 году.
  - е) Население Сан-Диего в 1990 году.
  - ж) Год, в который Роджер Уильямс основал г. Провиденс, штат Род-Айленд.
  - з) Высота горы Килиманджаро в футах.
  - и) Длина Бруклинского моста в футах.
  - к) Количество смертных случаев из-за автомобильных аварий в Соединенных Штатах в 1992 году.
- 16.2.** Билеты в лотерее стоят 1 доллар. Существуют два возможных приза: выигрыш в 10 долларов с вероятностью 1/50 и выигрыш в 1 000 000 долларов с вероятностью 1/2 000 000. Какова ожидаемая денежная стоимость лотерейного билета? Когда решение по приобретению такого билета является рациональным (если оно вообще бывает таким)? Дайте точный ответ — составьте уравнение, касающееся полезностей. Вы можете принять предположение, что ваш текущий капитал равен  $k$  долларов и что  $U(S_k) = 0$ . Вы можете также предпо-

ложить, что  $U(S_{k+10}) = 10 \times U(S_{k+1})$ , но не должны принимать никаких предложений в отношении полезности  $U(S_{k+1} \dots 000)$ . Социологические исследования показывают, что люди с низкими доходами покупают несоответствующее их благосостоянию количество лотерейных билетов. Как вы считаете, связано ли это с тем, что они не умеют принимать рациональные решения, или с тем, что они руководствуются другой функцией полезности?

- 16.3.** В 1738 году Даниил Бернули исследовал санкт-петербургский парадокс, который заключается в следующем. У вас есть возможность сыграть в игру, в которой подлинная монета подбрасывается повторно до тех пор, пока не выпадет орлом вверх. Если орел впервые появится на  $n$ -м броске, вы выигрываете  $2^n$  долларов.
- a) Покажите, что ожидаемое денежное значение этой игры является бесконечно большим.
  - б) Сколько бы вы лично заплатили за участие в этой игре?
  - в) Бернули разрешил кажущийся парадокс, связанный с нежеланием людей участвовать в этой игре, несмотря на ее привлекательность, выдвинув предположение, что полезность денег измеряется логарифмической шкалой (т.е.  $U(S_n) = a \log_2 n + b$ , где  $S_n$  — денежное состояние, связанное с наличием  $n$  долларов). Какова ожидаемая полезность этой игры согласно указанному предположению?
  - г) Каковая максимальная сумма, решение по уплате которой за участие в этой игре было бы рациональным, при условии, что первоначальное состояние рассматриваемого лица измеряется суммой  $k$  долларов?
- 16.4.** Определите вашу собственную оценку полезности различных постепенно увеличивающихся сумм денег, выполнив ряд проверок предпочтения между некоторой определенной суммой  $M_1$  и лотереей  $[p, M_2; (1-p), 0]$ . Выбирайте различные значения  $M_1$  и  $M_2$  и варьируйте вероятность  $p$  до тех пор, пока для вас эти два варианта не станут безразличными. Нанесите полученную в результате функцию полезности на график.
- 16.5.** Напишите компьютерную программу для автоматизации процесса, описанного в упр. 16.4. Проверьте работу вашей программы на нескольких людях с разным собственным капиталом и различными политическими взглядами. Прокомментируйте результаты сравнения согласованности полученных данных как для отдельного лица, так и для разных лиц.
- 16.6.** Какова стоимость микрошанса смерти для вас? Разработайте определенный протокол, позволяющий узнать это значение. Задавайте вопросы, основанные на том, сколько вы готовы заплатить, чтобы избежать риска смерти, и на том, сколько вы готовы принять в качестве платы за то, чтобы вы взяли на себя этот риск.
- 16.7.** Покажите, что если переменные  $X_1$  и  $X_2$  независимы по предпочтениям от переменной  $X_3$ , а  $X_2$  и  $X_3$  независимы по предпочтениям от  $X_1$ , то  $X_3$  и  $X_1$  независимы по предпочтениям от  $X_2$ .
- 16.8.** В этом и двух следующих упражнениях завершается анализ задачи выбора площадки для размещения аэропорта, приведенной на рис. 16.5.

- a) Приведите приемлемые данные об областях определения переменных, вероятностях и полезностях для этой сети, при условии, что имеются три возможных площадки.
- б) Решите эту задачу принятия решений.
- в) Что произойдет, если будут внедрены такие технологические усовершенствования, что каждый самолет станет вырабатывать вдвое меньше шума?
- г) А что произойдет, если требования по предотвращению шума станут в три раза более значимыми?
- д) Рассчитайте значение VPI для переменных вашей модели *AirTraffic* (Интенсивность воздушного движения), *Litigation* (Возможности урегулирования формальностей) и *Construction* (Условия строительства аэропорта).

**16.9.** Повторите упр. 16.8, используя представление “действие–полезность”, показанное на рис. 16.6.

**16.10.** К какому элементу таблицы условных вероятностей является наиболее чувствительным значение полезности на любой из схем выбора площадки для аэропорта из упр. 16.8 и 16.9, если даны все доступные свидетельства?

**16.11.** (*Адаптировано из [1191].*) Покупатель подержанного автомобиля может принять решение выполнить различные проверки с разной стоимостью (например, поступать по шинам, показать автомобиль квалифицированному механику), а затем, в зависимости от результата этих проверок, принять решение о том, какой автомобиль следует купить. Предполагается, что покупатель принимает решение о покупке автомобиля  $c_1$ , что он собирается провести самое большее одну проверку и что этой проверкой автомобиля  $c_1$  является  $t_1$ , которая стоит 50 долларов.

Автомобиль может находиться в хорошем состоянии (качество  $q^+$ ) или в плохом состоянии (качество  $q^-$ ), а проверка может показать, в каком состоянии находится автомобиль. Автомобиль  $c_1$  стоит 1500 долларов, а его рыночная стоимость равна 2000 долларов, если он находится в хорошем состоянии; если же нет, то потребуется ремонт стоимостью 700 долларов, чтобы привести его в хорошее состояние. Оценка покупателя состоит в том, что автомобиль  $c_1$  имеет 70% шансов на то, чтобы оказаться в хорошем состоянии.

- а) Нарисуйте сеть принятия решений, которая представляет данную задачу.
- б) Вычислите ожидаемую чистую прибыль от покупки автомобиля  $c_1$ , если не проводится проверка.
- в) Проверки можно описать с помощью оценки вероятности того, пройдет ли автомобиль или не пройдет данную конкретную проверку, позволяющую определить, находится ли автомобиль в хорошем или плохом состоянии. Имеется следующая информация:

$$\begin{aligned} P(\text{pass}(c_1, t_1) | q^+(c_1)) &= 0.8 \\ P(\text{pass}(c_1, t_1) | q^-(c_1)) &= 0.35 \end{aligned}$$

Примените теорему Байеса для вычисления вероятности успешного прохождения (или не прохождения) автомобилем проверки и следовательно, вероятности того, что он находится в хорошем (или плохом) состоянии, с учетом каждого возможного результата проверки.

- г) Рассчитайте оптимальные решения при условии прохождения или не прохождения проверки, а также их ожидаемые полезности.
- д) Рассчитайте стоимость информации о проверке и разработайте оптимальный условный план для покупателя.
- 16.12.** Докажите, что стоимость информации является неотрицательной и независимой от последовательности восприятий, как утверждалось в разделе 16.6. Объясните, как может случиться, что после получения информации будет принято худшее решение, чем было бы до ее получения.
- 16.13.** Модифицируйте и дополните программы байесовской сети, приведенные в репозитарии кода, чтобы обеспечить создание и оценку сетей принятия решений, а также вычисление стоимости информации.

**Ответы к упр. 16.1:**

- первый ряд вопросов: 3 000 000, 1 600 000, 1541, 41 000 000, 4768, 221, 649 000 000, 295 000 000, 132, 25 546;
- второй ряд вопросов: 1917, 155 000 000, 4 500 000, 11 000 000, 120 000, 1 100 000, 1 636, 19 340, 1 595, 41 710.

# 17 ПРИНЯТИЕ СЛОЖНЫХ РЕШЕНИЙ

*В данной главе рассматриваются методы принятия решений о том, что следует делать в настоящее время, при условии, что в дальнейшем может быть принято другое решение.*

В этой главе описано, какие расчеты связаны с принятием решений. В главе 16 речь шла о задачах принятия единоразовых или эпизодических решений, в которых полезность результата каждого действия была вполне известна, а в настоящей главе рассматриваются **задачи последовательного принятия решений**, в которых полезность действий агента зависит от последовательности решений. Задачи последовательного принятия решений, в которых рассматриваются полезности, степени неопределенности и результаты восприятия, являются обобщением задач поиска и планирования, описанных в частях II и IV. В разделе 17.1 описано, как должны быть определены задачи последовательного принятия решений, а в разделах 17.2 и 17.3 показано, как их следует решать, чтобы выработать оптимальные правила поведения, в которых уравновешиваются риски и вознаграждения, связанные с осуществлением действий в неопределенной среде. В разделе 17.4 эти идеи распространяются на случай частично наблюдаемых вариантов среды, а в разделе 17.5 разрабатывается полный проект для агентов, действующих на основе теории принятия решений в частично наблюдаемых вариантах среды; в этом проекте объединяются динамические байесовские сети, описанные в главе 15, и сети принятия решений, описанные в главе 16.

Во второй части данной главы рассматриваются варианты среды с многочисленными агентами. В таких вариантах среды понятие оптимального поведения становится гораздо более сложным из-за взаимодействия агентов. В разделе 17.6 представлены основные идеи **теории игр**, включая ту идею, что рациональным агентам может потребоваться вести себя случайным образом. В разделе 17.7 показано, как следует проектировать мультиагентные системы для того, чтобы несколько агентов могли достичь общей цели.

## 17.1. ЗАДАЧИ ПОСЛЕДОВАТЕЛЬНОГО ПРИНЯТИЯ РЕШЕНИЙ

### Пример

Предположим, что агент находится в среде с размерами  $4 \times 3$ , показанной на рис. 17.1, а. Начиная с начального состояния, он должен выбирать какое-то действие в каждом временном интервале. Взаимодействие со средой оканчивается после того, как агент достигает одного из целевых состояний, обозначенных  $+1$  и  $-1$ . В каждом местонахождении в распоряжении агента имеются действия *Up* (Вверх), *Down* (Вниз), *Left* (Влево) и *Right* (Вправо). На данный момент предполагается, что эта среда является **полностью наблюдаемой**, поэтому агент всегда знает, где он находится.

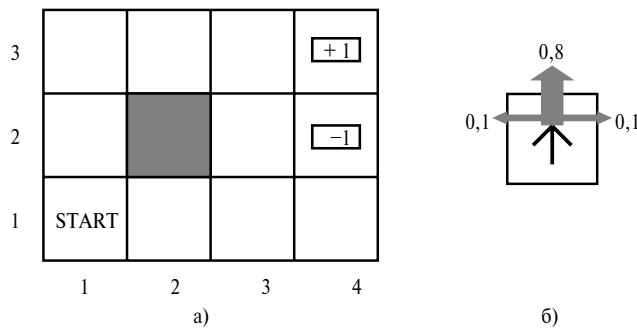


Рис. 17.1. Определение задачи: простая среда с размерами  $4 \times 3$ , в которой перед агентом поставлена задача последовательного принятия решений (а); модель перехода для этой среды: “намеченный” результат достигается с вероятностью 0,8, а с вероятностью 0,2 агент движется под прямыми углами влево или вправо от намеченного направления (б). Столкновение со стеной приводит к тому, что дальнейшее движение не происходит. С двумя конечными состояниями связаны вознаграждения  $+1$  и  $-1$  соответственно, а со всеми другими состояниями связано вознаграждение  $-0,04$

Если бы эта среда была полностью детерминированной, то достижение требуемого решения было бы несложным: [*Up*, *Up*, *Right*, *Right*, *Right*]. К сожалению, среда не всегда реагирует правильно на осуществление этого решения, поскольку действия выполняются ненадежно. Конкретная принятая нами модель стохастического движения показана на рис. 17.1, б. Каждое действие достигает намеченной цели с вероятностью 0,8, но в течение всего остального времени в результате выполнения действия агент движется под прямыми углами к выбранному направлению. Более того, если агент ударяется в стену, то остается в том же квадрате. Например, выполняя из начального квадрата (1, 1) действие *Up* перемещает агента в квадрат (1, 2) с вероятностью 0,8, но с вероятностью 0,1 агент движется вправо, в квадрат (2, 1), а с вероятностью 0,1 он движется влево, ударяется в стену и остается в квадрате (1, 1). В такой среде последовательность действий [*Up*, *Up*, *Right*, *Right*, *Right*] позволяет обойти барьер и достичь целевого состояния, квадрата (4, 3), с вероятностью

$0.8^5 = 0.32768$ . Существует также небольшой шанс случайно достичь цели, обойдя барьер с другой стороны с вероятностью  $0.1^4 \times 0.8$ , поэтому суммарная вероятность достижения цели равна  $0.32776$  (см. также упр. 17.1).

Спецификацию вероятностей результатов каждого действия в каждом возможном состоянии принято называть **моделью перехода** (или просто “моделью”, если не может возникнуть путаница). Для обозначения вероятности достижения состояния  $s'$ , если в состоянии  $s$  было выполнено действие  $a$ , будет применяться запись  $T(s, a, s')$ . Предполагается, что эти переходы являются **марковскими** в том смысле, какой указан в главе 15, т.е. что вероятность достижения состояния  $s'$  из  $s$  зависит только от  $s$ , а не от истории пребывания в предыдущих состояниях. На данный момент запись  $T(s, a, s')$  может рассматриваться как большая трехмерная таблица, содержащая вероятности. В дальнейшем, в разделе 17.5, будет показано, что модель перехода может быть представлена как **динамическая байесовская сеть**, точно так же, как и в главе 15.

В завершение этого определения среды задачи необходимо сформулировать функцию полезности для агента. Поскольку эта задача принятия решений является последовательной, функция полезности должна зависеть от последовательности состояний (от **истории пребывания в среде**), а не от отдельного состояния. Ниже в этом разделе будет приведено описание того, как такие функции полезности могут быть определены в целом, а на данный момент просто примем предположение, что в каждом состоянии  $s$  агент получает **вознаграждение**  $R(s)$ , которое может быть положительным или отрицательным, но должно быть ограниченным. В данном конкретном примере вознаграждение равно  $-0.04$  во всех состояниях, кроме конечных (с которыми связаны вознаграждения  $+1$  и  $-1$ ). Полезность, связанная с историей пребывания в среде (на данный момент), рассматривается как сумма полученных вознаграждений. Например, если агент достиг состояния  $+1$  после 10 шагов, суммарная полезность его действий будет равна  $0.6$ . Отрицательное вознаграждение  $-0.04$  побуждает агента быстрее достичь квадрата  $(4, 3)$ , поэтому данная среда представляет собой стохастическое обобщение вариантов среды, которые рассматривались в задачах поиска в главе 3. Еще один способ описать эту игровую ситуацию состоит в том, что агенту “не нравится” находиться в этой среде, поэтому он стремится выйти из игры как можно быстрее.

Такая спецификация задачи последовательного принятия решений для полностью наблюдаемой среды с марковской моделью перехода и дополнительными вознаграждениями называется спецификацией **марковского процесса принятия решений**, или сокращенно **MDP** (Markov Decision Process). Любая задача MDP определяется тремя перечисленными ниже компонентами.

- Начальное состояние —  $S_0$ .
- Модель перехода —  $T(s, a, s')$ .
- Функция вознаграждения<sup>1</sup> —  $R(s)$ .

---

<sup>1</sup> В некоторых определениях задач MDP допускается, чтобы вознаграждение зависело также от действия и результата, поэтому функция вознаграждения принимает вид  $R(s, a, s')$ . Такой подход позволяет упростить описание некоторых вариантов среды, но не приводит к какому-либо фундаментальному изменению самой задачи.

Следующий вопрос состоит в том, как должно выглядеть решение этой задачи. Выше в данной главе было показано, что какая-либо фиксированная последовательность действий не может служить решением этой задачи, поскольку в конечном итоге после ее выполнения агент может оказаться в состоянии, отличном от целевого. Поэтому в решении должно быть указано, что следует делать агенту в любом состоянии, которого он может достичь. Решение такого рода — это так называемая **стратегия**. Для обозначения стратегии обычно принято использовать  $\pi$ ; а  $\pi(s)$  — это действие, рекомендованное в соответствии со стратегией  $\pi$  для состояния  $s$ . Если агент имеет полное описание стратегии, то всегда знает, что делать дальше, независимо от результата любого действия.

Каждый раз, когда осуществляется данная конкретная стратегия, начиная с начального состояния, стохастический характер среды приводит к формированию другой истории пребывания в среде. Поэтому качество определения стратегии изменяется по ожидаемой полезности возможных историй пребывания в среде, создаваемых с помощью этой стратегии. **Оптимальной стратегией** называется такая стратегия, которая позволяет достичь максимальной ожидаемой полезности. Для обозначения оптимальной стратегии принято использовать запись  $\pi^*$ . Если агенту указана стратегия  $\pi^*$ , он принимает решение, что делать, проверяя свои текущие результаты восприятия, которые сообщают ему, что он находится в текущем состоянии  $s$ , а затем выполняя действие  $\pi^*(s)$ . В любой стратегии функция агента представлена явно, поэтому стратегия является описанием простого рефлексного агента, сформированным с учетом информации, которая используется агентом, действующим на основе полезности.

Оптимальная стратегия для мира, приведенного на рис. 17.1, показана на рис. 17.2, а. Обратите внимание на то, что стоимость выполнения одного шага довольно мала по сравнению со штрафом, который связан со случайным попаданием в квадрат  $(4, 2)$ , поэтому оптимальная стратегия для состояния  $(3, 1)$  является предельно осторожной. Этот стратегия рекомендует, что нужно совершить дальний обход препятствия, а не пытаться пройти по короткому пути и тем самым подвергнуться риску попасть в квадрат  $(4, 2)$ .

Равновесие между риском и вознаграждением изменяется в зависимости от значения функции  $R(s)$  для нетерминальных состояний. На рис. 17.2, б показаны оптимальные стратегии для четырех различных диапазонов изменения значения  $R(s)$ . Если  $R(s) \leq -1.6284$ , жизнь настолько мучительна, что агент направляется прямо к ближайшему выходу, даже если стоимость этого выхода равна  $-1$ . Если  $-0.4278 \leq R(s) \leq -0.0850$ , жизнь довольно дискомфортна; агент выбирает кратчайший маршрут к состоянию  $+1$  и стремится избежать риска случайного попадания в состояние  $-1$ . В частности, агент выбирает короткий путь из квадрата  $(3, 1)$ . А если жизнь не так уж неприятна ( $-0.0221 < R(s) < 0$ ), оптимальная стратегия состоит в том, чтобы избегать вообще какого-либо риска. В квадратах  $(4, 1)$  и  $(3, 2)$  агент направляется буквально прочь от состояния  $-1$ , чтобы случайно не попасть туда ни при каких обстоятельствах, даже несмотря на то, что из-за этого ему приходится несколько раз удариться головой о стену. Наконец, если  $R(s) > 0$ , то жизнь агента становится весьма приятной и он избегает обоих выходов. При условии, что используются действия, показанные в квадратах  $(4, 1)$ ,  $(3, 2)$  и  $(3, 3)$ , любая стратегия является оптимальной и агент получает бесконечно боль-

шое суммарное вознаграждение, поскольку он никогда не попадает в терминальное состояние. Как это ни удивительно, но оказывается, что существуют шесть других оптимальных стратегий для различных диапазонов значений  $R(s)$ ; в упр. 17.7 предлагаются найти эти стратегии.

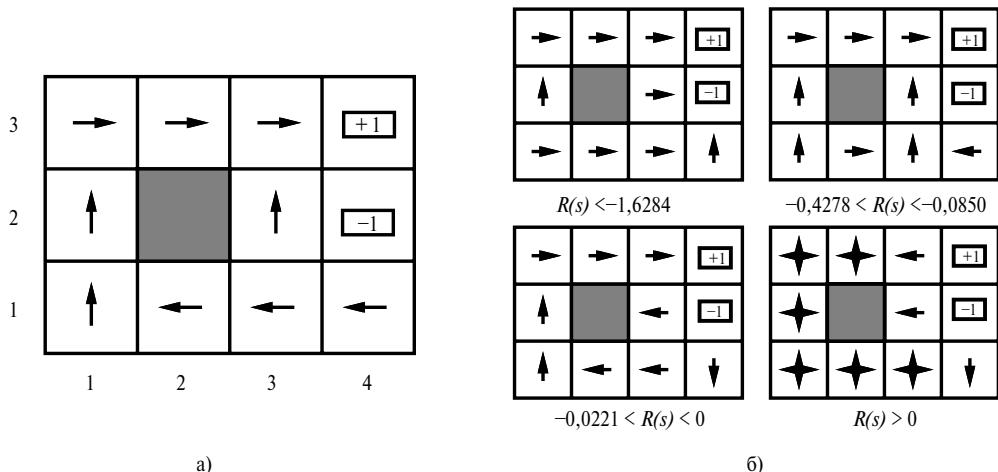


Рис. 17.2. Примеры оптимальных стратегий: оптимальная стратегия для стохастической среды со значениями  $R(s) = -0.04$  в нетерминальных состояниях (а); оптимальные стратегии для четырех различных диапазонов значений  $R(s)$  (б)

Тщательное уравновешивание риска и вознаграждения является характерной особенностью задач MDP, которая не возникает в детерминированных задачах поиска; более того, такое уравновешивание характерно для многих реальных задач принятия решений. По этой причине задачи MDP изучаются в нескольких научных областях, включая искусственный интеллект, исследование операций, экономику и теорию управления. Для вычисления оптимальных стратегий были предложены десятки алгоритмов. В разделах 17.2 и 17.3 описываются два наиболее важных семейства алгоритмов. Но вначале мы должны завершить начатое исследование полезностей и стратегий для задач последовательного принятия решений.

### Оптимальность в задачах последовательного принятия решений

В примере задачи MDP (см. рис. 17.1) производительность агента определялась по сумме вознаграждений, связанных с посещенными состояниями. Такой выбор показателя производительности нельзя назвать произвольным, но он не является также единствено допустимым. В данном разделе рассматриваются возможные варианты показателей производительности, т.е. варианты способов определения функции полезности по историям пребывания в среде, которые могут записываться как  $U_h([s_0, s_1, \dots, s_n])$ . Этот раздел основан на идеях, изложенных в главе 16, и является довольно формальным; основные его пункты подытожены в конце.

Первый вопрос, на который нужно найти ответ, состоит в том, существует ли **конечный горизонт** или **бесконечный горизонт** для принятия решений. Наличие конечного горизонта означает, что есть такое фиксированное время  $N$ , после кото-

рого все теряет смысл, — так сказать, игра все равно окончена. Таким образом,  $U_h([s_0, s_1, \dots, s_{N+k}]) = U_h([s_0, s_1, \dots, s_N])$  для всех  $k > 0$ . Например, предположим, что агент начинает свое движение с квадрата  $(3, 1)$  в мире с размерами  $4 \times 3$ , показанном на рис. 17.1, а также допустим, что  $N=3$ . В таком случае, чтобы получить хоть малейший шанс достичь состояния  $+1$ , агент должен направиться непосредственно к нему, и оптимальное действие состоит в том, чтобы двигаться в направлении  $Up$ . С другой стороны, если  $N=100$ , то запас времени настолько велик, что можно выбрать безопасный маршрут в направлении  $Left$ . *Поэтому при наличии конечного горизонта оптимальное действие в каждом конкретном состоянии со временем может измениться.* Принято считать, что оптимальная стратегия при наличии конечного горизонта является **нестационарной**. С другой стороны, если нет заданного предела времени, то нет смысла вести себя по-разному в одном и том же состоянии в разное время. Поэтому оптимальное действие зависит только от текущего состояния и оптимальная стратегия является **стационарной**. Таким образом, стратегии для случая с бесконечным горизонтом проще по сравнению с теми, которые применяются в случае с конечным горизонтом, и в данной главе будет в основном рассматриваться случай с бесконечным горизонтом<sup>2</sup>. Обратите внимание на то, что понятие “бесконечного горизонта” не обязательно означает, что все последовательности состояний являются бесконечными; оно просто говорит о том, что для их выполнения не устанавливаются фиксированные сроки. В частности, в любой задаче MDP с бесконечным горизонтом могут существовать конечные последовательности состояний, содержащие терминальное состояние.

Следующий вопрос, на который необходимо найти ответ, состоит в том, как рас-считать полезность последовательностей состояний. Мы будем рассматривать задачу поиска ответа на этот вопрос как задачу **многоатрибутной теории полезности** (см. раздел 16.4), где каждое состояние  $s_i$  рассматривается как атрибут последовательности состояний  $[s_0, s_1, s_2, \dots]$ . Чтобы получить простое выражение в терминах атрибутов, необходимо принять своего рода предположение о независимости предпочтений. Наиболее естественное предположение состоит в том, что отношение предпочтения агента между последовательностями состояний является **стационарным**. Стационарность предпочтений означает следующее: если две последовательности состояний,  $[s_0, s_1, s_2, \dots]$  и  $[s'_0, s'_1, s'_2, \dots]$ , начинаются с одного и того же состояния (т.е.  $s_0 = s'_0$ ), то эти две последовательности должны быть упорядочены по предпочтениям таким же образом, как и последовательности  $[s_1, s_2, \dots]$  и  $[s'_1, s'_2, \dots]$ . На естественном языке эту мысль можно выразить так, что если вы предпочитаете одно будущее развитие событий, начинающееся завтра, другому развитию событий, то вы должны также предпочесть это будущее развитие событий, если оно начнется сегодня. На первый взгляд, предположение о стационарности выглядит довольно безобидно, но влечет за собой весьма важные последствия: как оказалось, в условиях стационарности существуют только два способа присваивания значений полезности последовательностям, которые описаны ниже.

- 1. Аддитивные вознаграждения.** Полезность последовательности состояний определяется следующим образом:

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

---

<sup>2</sup> Это утверждение касается полностью наблюдаемых вариантов среды. Как будет показано ниже, для частично наблюдаемых вариантов среды случай с бесконечным горизонтом не так уж прост.

В мире  $4 \times 3$ , показанном на рис. 17.1, используются аддитивные вознаграждения. Обратите внимание на то, что свойство аддитивности уже было определено неявно в используемых нами функциях стоимости пути для алгоритмов эвристического поиска (см. главу 4).

2. **Обесцениваемые вознаграждения.** Полезность последовательности состояний определяется с помощью следующего соотношения:

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

где  $\gamma$  — это **коэффициент обесценивания**, который представляет собой число от 0 до 1. Коэффициент обесценивания описывает предпочтение агентом текущих вознаграждений перед будущими вознаграждениями. Если коэффициент  $\gamma$  близок к 0, вознаграждения, которые должны быть получены в отдаленном будущем, рассматриваются как малозначащие, а если коэффициент  $\gamma$  равен 1, то обесцениваемые вознаграждения полностью эквивалентны аддитивным вознаграждениям, поэтому аддитивные вознаграждения представляют собой частный случай обесцениваемых вознаграждений. По-видимому, обесценивание представляет собой хорошую модель изменения во времени предпочтений и животных, и человека. Коэффициент обесценивания  $\gamma$  эквивалентен процентной ставке  $(1/\gamma) - 1$ .

По причинам, которые вскоре станут очевидными, до конца этой главы предполагается, что используются обесцениваемые вознаграждения, хотя иногда допускается применение значения  $\gamma=1$ .

Сделанный нами выбор бесконечных горизонтов становится причиной возникновения определенной проблемы: если среда не содержит терминальное состояние или если агент никогда его не достигает, то все истории пребывания в среде будут иметь бесконечную длину, а полезности, связанные с аддитивными вознаграждениями, в общем случае будут бесконечными. Дело в том, что, безусловно,  $+\infty$  лучше, чем  $-\infty$ , но гораздо сложнее сравнивать две последовательности состояний, притом что обе имеют полезность  $+\infty$ . Для решения этой проблемы можно применить три описанных ниже подхода, два из которых уже упоминались в этой главе.

1. При наличии обесцениваемых вознаграждений полезность любой бесконечной последовательности является конечной. В действительности, если вознаграждения ограничены значением  $R_{\max}$  и  $\gamma < 1$ , то можно получить следующее соотношение с использованием стандартной формулы суммы бесконечных геометрических рядов:

$$U_h([s_0, s_1, s_2, \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max} / (1 - \gamma) \quad (17.1)$$

2. Если среда содержит терминальные состояния и гарантируется достижение агентом в конечном итоге одного из этих состояний, то нам никогда не придется сравнивать бесконечные последовательности действий. Стратегия, который гарантирует достижение терминального состояния, называется **правильной стратегией**. При наличии правильных стратегий можно использовать  $\gamma=1$  (т.е. аддитивные вознаграждения). Первые три стратегии, пока-

занные на рис. 17.2, б, являются правильными, а четвертая — неправильной. В ней достигается бесконечное суммарное вознаграждение за счет предотвращения попадания в терминальные состояния, притом что вознаграждение за пребывание в нетерминальных состояниях является положительным. Само существование неправильных стратегий в сочетании с использованием аддитивных вознаграждений может стать причиной неудачного завершения стандартных алгоритмов решения задач MDP, поэтому является весомым доводом в пользу применения обесцениваемых вознаграждений.

3. Еще один подход состоит в том, чтобы сравнивать бесконечные последовательности по **среднему вознаграждению**, получаемому в расчете на каждый временной интервал. Предположим, что с квадратом  $(1, 1)$  в мире  $4 \times 3$  связано вознаграждение  $0.1$ , тогда как для других нетерминальных состояний предусмотрено вознаграждение  $0.01$ . В таком случае стратегия, в которой агент предпочитет оставаться в квадрате  $(1, 1)$ , позволит получать более высокое среднее вознаграждение по сравнению с той стратегией, в которой агент находится в каком-то другом квадрате. Среднее вознаграждение представляет собой полезный критерий для некоторых задач, но анализ алгоритмов со средним вознаграждением выходит за рамки данной книги.

Подводя итог, можно сказать, что использование обесцениваемых вознаграждений связано с наименьшими трудностями при оценке последовательностей состояний. Заключительный этап состоит в том, чтобы показать, как осуществляется выбор между стратегиями с учетом того, что каждая конкретная стратегия  $\pi$  вырабатывает не только одну последовательность состояний, но целый ряд возможных последовательностей состояний, притом что каждая из этих последовательностей имеет конкретную вероятность, определяемую моделью перехода для данной среды. Таким образом, стоимость любой стратегии представляет собой ожидаемую сумму полученных обесцениваемых вознаграждений, где это ожидаемое значение вычисляется по всем возможным последовательностям состояний, которые могут возникнуть при осуществлении данной стратегии. Любая оптимальная стратегия  $\pi^*$  удовлетворяет следующему соотношению:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right] \quad (17.2)$$

В следующих двух разделах описаны алгоритмы поиска оптимальных стратегий.

## 17.2. ИТЕРАЦИЯ ПО ЗНАЧЕНИЯМ

В этом разделе представлен алгоритм вычисления оптимальной стратегии, называемый **итерацией по значениям**. Основная его идея состоит в том, что нужно рассчитать полезность каждого состояния, а затем использовать полезности состояний для выбора оптимального действия в каждом состоянии.

## Полезности состояний

Полезность состояний определяется в терминах полезности последовательностей состояний. Грубо говоря, полезность любого состояния представляет собой ожидаемую полезность последовательностей состояний, которые могут привести к этому состоянию. Очевидно, что перечень таких последовательностей состояний зависит от осуществляемой стратегии, поэтому начнем с определения полезности  $U^\pi(s)$  по отношению к конкретной стратегии  $\pi$ . Если мы предположим, что  $s_t$  — это состояние, в котором находится агент после осуществления стратегии  $\pi$  в течение  $t$  шагов (обратите внимание на то, что  $s_t$  — случайная переменная), то получим следующее:

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0=s \right] \quad (17.3)$$

На основании этого определения можно утверждать, что истинная полезность любого состояния, которую обозначим как  $U(s)$ , представляет собой  $U^{\pi^*}(s)$ , т.е. ожидаемую сумму обесцениваемых вознаграждений, при условии, что агент осуществляет оптимальную стратегию. Обратите внимание на то, что  $U(s)$  и  $R(s)$  — совершенно разные величины;  $R(s)$  — это “кратковременное” вознаграждение за пребывание в состоянии  $s$ ;  $U(s)$  — “долговременное” суммарное вознаграждение, которое начинается с состояния  $s$  и продолжается дальше. На рис. 17.3 показаны рассматриваемые значения полезности для мира  $4 \times 3$ . Заслуживает внимание то, что значения полезности по мере приближения состояний к выходу +1 становятся выше, поскольку уменьшается количество шагов, требуемых для достижения этого выхода.

3	0,812	0,868	0,918
2	0,762		0,660
1	0,705	0,655	0,611
	1	2	3
			4

Рис. 17.3. Полезности состояний в мире  $4 \times 3$ , рассчитанные при  $\gamma=1$  и  $R(s)=-0.04$  для нетерминальных состояний

Эта функция полезности  $U(s)$  позволяет агенту выбирать действия с использованием принципа максимальной ожидаемой полезности, приведенного в главе 16, т.е. выбирать действие, которое максимизирует ожидаемую полезность в следующем состоянии:

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s') U(s') \quad (17.4)$$

Итак, если полезность некоторого состояния представляет собой ожидаемую сумму обесцениваемых вознаграждений, начиная с данного момента и дальше, то существует прямая связь между полезностью состояния и полезностью его соседних состояний: *полезность некоторого состояния равна сумме непосредственного вознаграждения за пребывание в этом состоянии и ожидаемой обесцениваемой полезности следующего состояния, при условии, что агент выбирает оптимальное действие.* Это означает, что полезность любого состояния можно определить с помощью следующего соотношения:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad (17.5)$$

Уравнение 17.5 называется *уравнением Беллмана* в честь Ричарда Беллмана [97]. Полезности состояний (определенные с помощью уравнения 17.3 как ожидаемые полезности дальнейших последовательностей состояний) являются решениями множества уравнений Беллмана. В действительности, как будет показано в следующих двух разделах, они являются уникальными решениями.

Рассмотрим одно из уравнений Беллмана для мира  $4 \times 3$ . Уравнение для состояния  $(1, 1)$  приведено ниже.

$$\begin{aligned} U(1,1) = & -0.04 + \gamma \max \{ & 0.8 U(1,2) + 0.1 U(2,1) + 0.1 U(1,1), & (Up) \\ & 0.9 U(1,1) + 0.1 U(1,2), & (Left) \\ & 0.9 U(1,1) + 0.1 U(2,1), & (Down) \\ & 0.8 U(2,1) + 0.1 U(1,2) + 0.1 U(1,1) \} & (Right) \end{aligned}$$

После подстановки в это уравнение чисел, приведенных на рис. 17.3, можно обнаружить, что наилучшим действием является *Up*.

### Алгоритм итерации по значениям

Уравнение Беллмана является основой алгоритма итерации по значениям, применяемого для решения задач MDP. Если существует  $n$  возможных состояний, то количество уравнений Беллмана также равно  $n$ , по одному для каждого состояния. Эти  $n$  уравнений содержат  $n$  неизвестных — полезностей состояний. Поэтому можно было бы заняться поиском решений системы этих уравнений, чтобы определить полезности. Тем не менее возникает одна проблема, связанная с тем, что эти уравнения являются нелинейными, поскольку оператор “*max*” — это нелинейный оператор. Системы линейных уравнений могут быть решены очень быстро с использованием методов линейной алгебры, а для решения систем нелинейных уравнений необходимо преодолеть некоторые проблемы. Один из возможных подходов состоит в использовании итерационных методов. Для этого нужно начать с произвольных исходных значений полезностей, вычислить правую часть уравнения и подставить ее в левую, тем самым обновляя значение полезности каждого состояния с учетом полезностей его соседних состояний. Такая операция повторяется до тех пор, пока не достигается равновесие. Допустим, что  $U_1(s)$  — это значение полезности для со-

стояния  $s$  в  $i$ -й итерации. Шаг итерации, называемый **обновлением Беллмана**, выглядит следующим образом:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \quad (17.6)$$

Если обновление Беллмана используется неопределенно большое количество раз, то гарантируется достижение равновесия (см. следующий подраздел), и в этом случае конечные значения полезности должны представлять собой решения уравнений Беллмана. В действительности они также представляют собой уникальные решения, и соответствующая стратегия (полученная с помощью уравнения 17.4) является оптимальной. Применяемый при этом алгоритм, называемый **Value-Iteration**, показан в листинге 17.1.

**Листинг 17.1. Алгоритм итерации по значениям для вычисления полезностей состояний. Условие завершения работы взято из уравнения 17.8**

---

```

function Value-Iteration(mdp,  $\epsilon$ ) returns функция полезности
  inputs: mdp, задача MDP с состояниями  $S$ , моделью перехода  $T$ ,
            функцией вознаграждения  $R$ , коэффициентом
            обесценивания  $\gamma$ 
             $\epsilon$ , максимально допустимая ошибка определения полезности
            любого состояния
  local variables:  $U$ ,  $U'$ , векторы полезностей для состояний из  $S$ ,
                     первоначально равные нулю
             $\delta$ , максимальное изменение полезности любого
            состояния во время итерации

  repeat
     $U \leftarrow U'$ ;  $\delta \leftarrow 0$ 
    for each состояние  $s$  in  $S$  do
       $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s') U[s']$ 
      if  $|U'[s] - U[s]| > \delta$  then  $\delta \leftarrow |U'[s] - U[s]|$ 
    until  $\delta < \epsilon(1-\gamma)/\gamma$ 
  return  $U$ 

```

---

Мы можем применить алгоритм итерации по значениям к миру  $4 \times 3$  (см. рис. 17.1, а). Начиная с исходных значений, равных нулю, полезности изменяются, как показано на рис. 17.4, а. Обратите внимание на то, как состояния, находящиеся на различных расстояниях от квадрата  $(4, 3)$ , накапливают отрицательное вознаграждение до тех пор, пока в какой-то момент не обнаруживается путь к состоянию  $(4, 3)$ , после чего значения полезности начинают возрастать. Алгоритм итерации по значениям может рассматриваться как способ распространения информации через пространство состояний с помощью локальных обновлений.

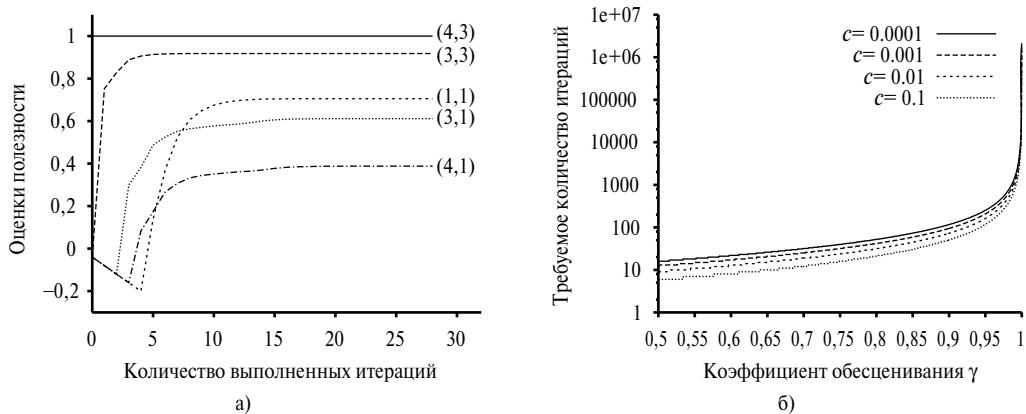


Рис. 17.4. Пример применения алгоритма итерации по значениям: график, показывающий изменение полезностей выбранных состояний в процессе итерации по значениям (а); количество итераций по значениям  $\kappa$ , необходимое для того, чтобы можно было гарантировать, что ошибка не превышает  $\varepsilon=c \cdot R_{\max}$  для различных значений  $c$ , как функция от коэффициента обесценивания  $\gamma$  (б)

### Сходимость итерации по значениям

Выше было указано, что процедура итерации по значениям в конечном итоге сходится к уникальному множеству решений уравнений Беллмана. В этом разделе показано, почему это происходит. В ходе этого будут представлены некоторые полезные математические идеи и получены определенные методы оценки ошибки в значении функции полезности, возвращаемом при преждевременном завершении работы алгоритма; это важно, поскольку означает, что количество применяемых итераций алгоритма не обязательно должно стремиться к бесконечности. Изложение в этом разделе является весьма формальным.

Основным понятием, используемым при доказательстве того, что процедура итерации по значениям сходится, является **сжатие**. Грубо говоря, функция сжатия — это функция от одного параметра, которая после ее последовательного применения к двум различным входным значениям вырабатывает два выходных значения, которые “ближе друг к другу” по меньшей мере на некоторую постоянную величину, чем первоначальные фактические параметры. Например, функция “деления на два” представляет собой функцию сжатия, поскольку после деления двух чисел на два разница между ними уменьшается наполовину. Обратите внимание на то, что функция “деления на два” имеет фиксированную точку, а именно нуль, которая остается неизменной в результате применения этой функции. На основании данного примера можно установить два важных свойства функций сжатия, описанных ниже.

- Функция сжатия имеет только одну фиксированную точку; если бы были две фиксированные точки, они бы не приближались друг к другу после применения функции, поэтому такая функция не соответствовала бы определению функции сжатия.
- После применения функции к любому параметру полученное значение должно стать ближе к фиксированной точке (поскольку фиксированная точка не

движется), поэтому в пределе повторное применение функции сжатия всегда приводит к достижению фиксированной точки.

Теперь предположим, что обновление Беллмана (уравнение 17.6) рассматривается как оператор  $B$ , который используется для одновременного обновления значений полезности каждого состояния. Обозначим через  $U_i$  вектор полезностей для всех состояний в  $i$ -й итерации. В таком случае уравнение обновления Беллмана может быть записано следующим образом:

$$U_{i+1} \leftarrow BU_i$$

Затем нужно найти способ измерения расстояний между векторами полезностей. Мы будем использовать **нормализованный максимум**, который измеряет длину вектора по длине его максимального компонента, следующим образом:

$$\|U\| = \max_s |U(s)|$$

При использовании этого определения “расстояние” между двумя векторами,  $\|U - U'\|$ , представляет собой максимальную разность между двумя соответствующими элементами. Основным математическим результатом данного раздела является такое утверждение: **допустим, что  $U_i$  и  $U_i'$  — два вектора полезностей. В таком случае получим следующее:**

$$\|BU_i - BU_i'\| \leq \gamma \|U_i - U_i'\| \quad (17.7)$$

*Это означает, что обновление Беллмана представляет собой функцию сжатия на коэффициент  $\gamma$ , применяемую к пространству векторов полезностей.* Таким образом, процедура итерации по значениям всегда сходится к уникальному решению уравнений Беллмана.

В частности, можно заменить значение  $U_i'$  в уравнении 17.7 истинными полезностями  $U$ , для которых  $BU=U$ . В таком случае будет получено следующее неравенство:

$$\|BU_i - U\| \leq \gamma \|U_i - U\|$$

Поэтому, если  $\|U_i - U\|$  рассматривается как ошибка в оценке  $U_i$ , то можно видеть, что эта ошибка уменьшается при каждой итерации на коэффициент, по меньшей мере равный  $\gamma$ . Это означает, что процедура итерации по значениям сходится экспоненциально быстро. Можно легко рассчитать количество итераций, требуемых для достижения заданной предельной ошибки  $\epsilon$ , как описано ниже. Вначале напомним, что, как показывает уравнение 17.1, полезности всех состояний ограничены значением  $\pm R_{\max} / (1-\gamma)$ . Из этого следует, что максимальная начальная ошибка определяется соотношением  $\|U_0 - U\| \leq 2R_{\max} / (1-\gamma)$ . Предположим, что для достижения ошибки, не превышающей  $\epsilon$ , выполнено  $N$  итераций. В таком случае потребуется  $\gamma^N \cdot 2R_{\max} / (1-\gamma) \leq \epsilon$  итераций, поскольку ошибка уменьшается каждый раз по меньшей мере на величину  $\gamma$ . Взяв логарифмы от этого выражения, можно определить, что достаточно применить следующее количество итераций:

$$N = \lceil \log(2R_{\max}/\epsilon(1-\gamma)) / \log(1/\gamma) \rceil$$

На рис. 17.4, б показано, как количество итераций  $N$  изменяется в зависимости от  $\gamma$  при различных значениях отношения  $\epsilon / R_{\max}$ . Положительной особенностью этого соотношения является то, что из-за экспоненциально быстрой сходимости значение

$N$  не очень зависит от отношения  $\epsilon / R_{\max}$ , а отрицательной особенностью — то, что  $N$  быстро возрастает по мере приближения значения  $\gamma$  к 1. Уменьшение значения  $\gamma$  позволяет добиться ускорения сходимости, но это фактически приводит к сужению горизонта агента и может не позволить агенту обнаруживать долговременные последствия своих действий.

Анализ предельной ошибки, приведенный выше, позволяет получить определенное представление о том, какие факторы влияют на продолжительность прогона данного алгоритма, но сам подход, основанный на определении предельной ошибки, иногда становится слишком консервативным способом принятия решения о прекращении итераций. Для последней цели можно использовать предел, связывающий ошибку с размерами обновления Беллмана в каждой конкретной итерации. На основании свойства сжатия (уравнение 17.7) можно показать, что если обновление невелико (т.е. не происходит значительного изменения полезности ни одного состояния), то ошибка также является небольшой по сравнению с истинным значением функции полезности. Точнее, выполняется следующее условие:

$$\text{if } ||U_{i+1} - U_i|| < \epsilon(1-\gamma) / \gamma \text{ then } ||U_{i+1} - U|| < \epsilon \quad (17.8)$$

В этом и состоит условие завершения, используемое в алгоритме `Value-Iteration`, который приведен в листинге 17.1.

До сих пор мы анализировали ошибку в значении функции полезности, возвращаемом алгоритмом итерации по значениям. ~~Но для агента фактически гораздо важнее то, насколько успешно он будет действовать, принимая свои решения на основе данной функции полезности.~~ Предположим, что после  $i$  итераций в процедуре итерации по значениям агент получает оценку  $U_i$  истинной полезности  $U$  и определяет максимальную ожидаемую полезность стратегии  $\pi_i$  на основе прогнозирования на один шаг вперед с использованием значения  $U_i$  (как в уравнении 17.4). Будет ли выбранное в итоге поведение почти столь же хорошим, как и оптимальное поведение? Это — крайне важный вопрос для любого реального агента, и было показано, что ответ на него является положительным. Значение  $U^{\pi_i}(s)$  — это полезность, достигаемая, если, начиная с состояния  $s$ , осуществляется стратегия  $\pi_i$ , а ~~убыточность стратегии~~  $||U^{\pi_i} - U||$  — это самая большая часть полезности, которую агент может потерять, осуществляя стратегию  $\pi_i$  вместо оптимальной стратегии  $\pi^*$ . Убыточность стратегии  $\pi_i$  связана с ошибкой в значении полезности  $U_i$  следующим неравенством:

$$\text{if } ||U_i - U|| < \epsilon \text{ then } ||U^{\pi_i} - U|| < 2\epsilon\gamma / (1-\gamma) \quad (17.9)$$

На практике часто происходит так, что стратегия  $\pi_i$  становится оптимальной задолго до того, как сходится значение  $U_i$ . На рис. 17.5 показано, как максимальная ошибка в значении  $U_i$  и убыточность стратегии приближаются к нулю по мере осуществления процедуры итерации по значениям для среды  $4 \times 3$  со значением  $\gamma = 0.9$ . Стратегия  $\pi_i$  становится оптимальной при  $i=4$ , даже несмотря на то, что максимальная ошибка в значении  $U_i$  все еще остается равной 0.46.

Теперь подготовлено все необходимое для использования процедуры итерации по значениям на практике. Известно, что процедура итерации по значениям в пределе сходится к правильным значениям полезности; ошибка в оценках полезностей может быть ограничена, даже если процедура итерации по значениям останавливается после конечного количества итераций; кроме того, может быть ограничена убы-

точность стратегии, которая связана с осуществлением соответствующей стратегии с максимальной ожидаемой полезностью. В качестве заключительного замечания отметим, что все результаты, приведенные в данном разделе, соответствуют такому случаю, когда применяется обесценивание полезностей, а  $\gamma < 1$ . Если  $\gamma = 1$  и среда содержит терминальные состояния, то можно вывести аналогичное множество результатов оценки сходимости и определения предельных значений ошибок, если выполняются некоторые формальные условия.

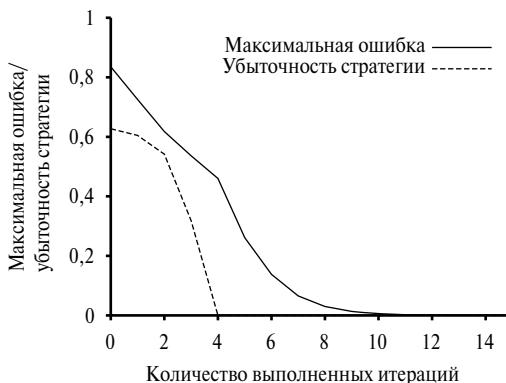


Рис. 17.5. Зависимости максимальной ошибки  $/|U_i - U|/$  в оценках полезности и убыточности стратегии  $/|U^{\pi_i} - U|/$  по сравнению с оптимальной стратегией от количества итераций в процедуре итерации по значениям

### 17.3. ИТЕРАЦИЯ ПО СТРАТЕГИЯМ

В предыдущем разделе было показано, что возможно выработать оптимальную стратегию, даже если оценка функции полезности является неточной. Если очевидно, что одно действие лучше по сравнению со всеми остальными, то нет необходимости точно определять истинные значения величины полезности всех рассматриваемых состояний. Эта идея подсказывает альтернативный метод поиска оптимальных стратегий. В алгоритме **итерации по стратегиям** чередуются описанные ниже два этапа, начиная с некоторой исходной стратегии  $\pi_0$ .

- **Оценка стратегии.** На основе стратегии  $\pi_i$  вычислить  $U_i = U^{\pi_i}$ , полезность каждого состояния, если будет осуществлена стратегия  $\pi_i$ .
- **Усовершенствование стратегии.** Вычислить новую стратегию  $\pi_{i+1}$  с максимальной ожидаемой полезностью, используя прогноз на один шаг и исходя из значения  $U_i$  (как в уравнении 17.4).

Алгоритм завершает свою работу после того, как этап усовершенствования стратегии не приводит к изменению значений полезности. Как известно, в этот момент функция полезности  $U_i$  представляет собой фиксированную точку обновления Беллмана, поэтому является решением уравнений Беллмана, а  $\pi_i$  должна быть оп-

тимальной стратегией. Поскольку для каждого конечного пространства состояний количество возможных стратегий является конечным и можно показать, что каждая итерация приводит к определению лучшей стратегии, то алгоритм итерации по стратегиям должен всегда завершать свою работу. Этот алгоритм показан в листинге 17.2.

#### Листинг 17.2. Алгоритм итерации по стратегиям для вычисления оптимальной стратегии

---

```

function Policy-Iteration(mdp) returns стратегия
    inputs: mdp, задача MDP с состояниями S, моделью перехода T
    local variables: U, U', векторы полезностей для состояний из S,
        первоначально равные нулю
        π, вектор стратегий, индексированный по состояниям,
        первоначально сформированный случайным образом

    repeat
        U  $\leftarrow$  Policy-Evaluation(π, U, mdp)
        unchanged?  $\leftarrow$  истинное значение
        for each состояние s in S do
            if  $\max_a \sum_{s'} T(s, a, s') U[s'] > \sum_{s'} T(s, \pi[s], s') U[s']$  then
                π[s]  $\leftarrow \operatorname{argmax}_a \sum_{s'} T(s, a, s') U[s']$ 
            unchanged?  $\leftarrow$  ложное значение
        until unchanged?
    return π

```

---

Очевидно, что этап усовершенствования стратегии является несложным, а как реализовать процедуру Policy-Evaluation? Оказалось, что осуществление такого подхода намного проще по сравнению с решением стандартных уравнений Беллмана (а именно это происходит в алгоритме итерации по значениям), поскольку действие, применяемое в каждом состоянии, зафиксировано в соответствии с выбранной стратегией. Стратегия  $\pi_i$  определяет действие  $\pi_i(s)$ , выполняемое в состоянии *s* на *i*-й итерации. Это означает, что можно воспользоваться упрощенной версией уравнения Беллмана (17.5), которая связывает полезность состояния *s* (соответствующую стратегии  $\pi_i$ ) с полезностями его соседних состояний, следующим образом:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \quad (17.10)$$

Например, предположим, что  $\pi_i$  — это стратегия, показанная на рис. 17.2, *a*. В таком случае имеет место  $\pi_i(1, 1) = U_2$ ,  $\pi_i(1, 2) = U_2$  и т.д., а упрощенные уравнения Беллмана принимают следующий вид:

$$\begin{aligned} U_i(1, 1) &= -0.04 + 0.8 U_i(1, 2) + 0.1 U_i(1, 1) + 0.1 U_i(2, 1) \\ U_i(1, 2) &= -0.04 + 0.8 U_i(1, 3) + 0.2 U_i(1, 2) \\ &\dots \end{aligned}$$

Важно отметить, что эти уравнения — линейные, поскольку оператор “*max*” был удален. Для *n* состояний имеется *n* линейных уравнений с *n* неизвестными, которые

могут быть решены точно за время  $O(n^3)$  с помощью стандартных методов линейной алгебры.

Для небольших пространств состояний оценка стратегии с использованием точных методов решения часто является наиболее эффективным подходом, а для больших пространств состояний затраты времени  $O(n^3)$  могут оказаться чрезмерно большими. К счастью, точная оценка стратегии не требуется. Вместо этого можно выполнить некоторое количество упрощенных этапов итерации по значениям (они являются упрощенными, поскольку стратегия зафиксирована) для получения достаточно хорошей аппроксимации полезности. Упрощенное обновление Беллмана для этого процесса определяется таким соотношением:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

и определяемая в нем операция подстановки повторяется  $k$  раз для получения следующей оценки полезности. Результатирующий алгоритм называется **модифицированной итерацией по стратегиям**. Он часто оказывается намного более эффективным, чем стандартная итерация по стратегиям или итерация по значениям.

Алгоритмы, описанные до сих пор в данной главе, требуют одновременного обновления полезности или стратегии для всех состояний. Как оказалось, применение такой организации работы не является строго необходимым. В действительности в каждой итерации можно выбирать любое подмножество состояний и применять к этому подмножеству либо тот, либо другой вид обновления (усовершенствование стратегии или упрощенную итерацию по значениям). Такой наиболее общий алгоритм называется **асинхронной итерацией по стратегиям**. При соблюдении определенных условий выбора исходной стратегии и функции полезности гарантируется сходимость асинхронной итерации по стратегиям к определенной оптимальной стратегии. А то, что мы вправе выбирать для работы с ними любые состояния, означает, что могут быть разработаны гораздо более эффективные эвристические алгоритмы, например, алгоритмы, которые сосредоточиваются на обновлении значений состояний, которые с наибольшей вероятностью будут достигнуты при осуществлении качественной стратегии. Такой подход имеет гораздо больше смысла в реальной жизни — если человек не намеревается попасть на прибрежную полосу, спрыгнув с высокой скалы, то для него нет смысла заниматься точной оценкой стоимости связанных с этим результатающих состояний.

## 17.4. МАРКОВСКИЕ ПРОЦЕССЫ ПРИНЯТИЯ РЕШЕНИЙ В ЧАСТИЧНО НАБЛЮДАЕМЫХ ВАРИАНТАХ СРЕДЫ

В описании марковских процессов принятия решений, приведенном в разделе 17.1, предполагалось, что среда является **полностью наблюдаемой**. При использовании этого предположения агент всегда знает, в каком состоянии он находится. Это предположение, в сочетании с предположением о марковости модели перехода, означает, что оптимальная стратегия зависит только от текущего состояния. А если среда является только **частично наблюдаемой**, то вполне очевидно, что ситуация становится гораздо менее ясной. Агент не всегда точно знает, в каком состоянии находится, поэтому не

может выполнить действие  $\pi(s)$ , рекомендуемое для этого состояния. Кроме того, полезность состояния  $s$  и оптимальное действие в состоянии  $s$  зависят не только от  $s$ , но и от того, насколько много агент знает, находясь в состоянии  $s$ . По этим причинам задачи **MDP в частично наблюдаемой среде** (Partially Observable MDP — POMDP, читается как “пом-ди-пи”) обычно рассматриваются как намного более сложные по сравнению с обычными задачами MDP. Однако невозможно игнорировать необходимость решения задач POMDP, поскольку реальный мир изобилует такими задачами.

В качестве примера еще раз рассмотрим мир с размерами  $4 \times 3$  (см. рис. 17.1), но на этот раз предположим, что агент вообще не имеет датчиков, а также не представляет себе, где находится. Точнее, допустим, что начальным состоянием агента с равной вероятностью может быть любое из девяти нетерминальных состояний (рис. 17.6, *a*). Очевидно, что если бы агент знал, что он находится в квадрате  $(3, 3)$ , то отправился бы направо, выполнив движение *Right*, а если бы знал, что он — в квадрате  $(1, 1)$ , то направился бы вверх с помощью движения *Up*, но поскольку агент может находиться в любом квадрате, то что ему делать? Один из возможных ответов состоит в том, что агенту необходимо вначале действовать так, чтобы уменьшить неопределенность своего положения и только после этого отправиться к выходу  $+1$ . Например, если агент выполнит движение *Left* пять раз, то с наибольшей вероятностью окажется у левой стены (см. рис. 17.6, *b*), а если он после этого пять раз выполнит движение *Up*, то, вполне вероятно, будет находиться вверху, возможно даже, в левом верхнем углу (см. рис. 17.6, *c*). Наконец, если он пять раз выполнит движение *Right*, то получит хорошие шансы (около 77,5%) достижения выхода  $+1$  (см. рис. 17.6, *d*). Продолжение после этого движения вправо повышает его шансы до 81,8%. Поэтому такая стратегия является удивительно безопасной, но при ее использовании агенту потребуется довольно много времени для достижения желаемого выхода, а ожидаемая полезность будет составлять лишь около 0.08. Оптимальная стратегия, который вскоре будет описана, позволяет достичь намного лучших результатов.

<table border="1"> <tr><td>0,111</td><td>0,111</td><td>0,111</td><td>0,000</td></tr> <tr><td>0,111</td><td></td><td>0,111</td><td>0,000</td></tr> <tr><td>0,111</td><td>0,111</td><td>0,111</td><td>0,111</td></tr> </table>	0,111	0,111	0,111	0,000	0,111		0,111	0,000	0,111	0,111	0,111	0,111	<table border="1"> <tr><td>0,300</td><td>0,010</td><td>0,008</td><td>0,000</td></tr> <tr><td>0,221</td><td></td><td>0,059</td><td>0,012</td></tr> <tr><td>0,371</td><td>0,012</td><td>0,008</td><td>0,000</td></tr> </table>	0,300	0,010	0,008	0,000	0,221		0,059	0,012	0,371	0,012	0,008	0,000	<table border="1"> <tr><td>0,622</td><td>0,221</td><td>0,071</td><td>0,024</td></tr> <tr><td>0,005</td><td></td><td>0,003</td><td>0,022</td></tr> <tr><td>0,003</td><td>0,024</td><td>0,003</td><td>0,000</td></tr> </table>	0,622	0,221	0,071	0,024	0,005		0,003	0,022	0,003	0,024	0,003	0,000	<table border="1"> <tr><td>0,005</td><td>0,007</td><td>0,019</td><td>0,775</td></tr> <tr><td>0,034</td><td></td><td>0,007</td><td>0,105</td></tr> <tr><td>0,005</td><td>0,006</td><td>0,008</td><td>0,030</td></tr> </table>	0,005	0,007	0,019	0,775	0,034		0,007	0,105	0,005	0,006	0,008	0,030
0,111	0,111	0,111	0,000																																																
0,111		0,111	0,000																																																
0,111	0,111	0,111	0,111																																																
0,300	0,010	0,008	0,000																																																
0,221		0,059	0,012																																																
0,371	0,012	0,008	0,000																																																
0,622	0,221	0,071	0,024																																																
0,005		0,003	0,022																																																
0,003	0,024	0,003	0,000																																																
0,005	0,007	0,019	0,775																																																
0,034		0,007	0,105																																																
0,005	0,006	0,008	0,030																																																
a)	б)	в)	г)																																																

Рис. 17.6. Пример реализации стратегии агента, не основанной на оптимальном подходе: первоначальное распределение вероятностей для местонахождения агента (*a*); распределение вероятностей после выполнения движения *Left* пять раз (*б*); то же, после выполнения движения *Up* пять раз (*в*); то же, после выполнения движения *Right* пять раз (*г*)

Чтобы найти подход к решению задач POMDP, необходимо вначале определить их должным образом. Любая задача POMDP состоит из таких же компонентов, как и задача MDP (модель перехода  $T(s, a, s')$  и функция вознаграждения  $R(s)$ ), но имеет также **модель наблюдения**  $O(s, o)$ , которая задает вероятность получения

результатов наблюдения  $o$  в состоянии<sup>3</sup>  $s$ . Например, рассматриваемый в данном случае агент без датчиков имеет только одно возможное наблюдение (пустое наблюдение), и такое наблюдение возникает с вероятностью 1 в любом состоянии.

В главах 3 и 12 рассматривались задачи планирования в недетерминированных и частично наблюдаемых вариантах среды и было определено **доверительное состояние** (множество фактических состояний, в которых может находиться агент) как ключевая концепция для описания и вычисления решений. В задачах POMDP это понятие требует определенного уточнения. Теперь доверительным состоянием  $b$  становится распределение вероятностей по всем возможным состояниям. Например, начальное доверительное состояние в ситуации, показанной на рис. 17.6,  $a$ , может быть записано как

$$\left\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \right\rangle.$$

Мы будем обозначать через  $b(s)$  вероятность, присвоенную фактическому состоянию  $s$  доверительным состоянием  $b$ . Агент может вычислить свое текущее доверительное состояние как распределение условных вероятностей по фактическим состояниям, если дана последовательность произошедших до сих пор наблюдений и действий. Такая задача по сути сводится к задаче **фильтрации** (см. главу 15). Основное рекурсивное уравнение фильтрации (см. уравнение 15.3) показывает, как вычислить новое доверительное состояние из предыдущего доверительного состояния и нового наблюдения. Для решения задач POMDP необходимо также учитывать определенное действие и использовать немного другую систему обозначений, но результат остается по сути тем же самым. Если предыдущим доверительным состоянием было  $b(s)$ , а агент выполнил действие  $a$  и получил результаты наблюдения  $o$ , то новое доверительное состояние определяется следующим соотношением:

$$b'(s') = \alpha \cdot O(s', o) \sum_s T(s, a, s') \cdot b(s) \quad (17.11)$$

где  $\alpha$  — нормализующая константа, при использовании которой сумма вероятностей доверительных состояний становится равной 1. Это уравнение можно сокращенно записать как  $b' = \text{Forward}(b, a, o)$ .

Основная идея, необходимая для понимания сути задач POMDP, состоит в следующем: ~~optимальное действие зависит только от текущего доверительного состояния агента~~. Это означает, что оптимальную стратегию можно описать, отобразив стратегию  $\pi^*(b)$  с доверительных состояний на действия. Она не зависит от фактического состояния, в котором находится агент. Это — очень благоприятная особенность, поскольку агент не знает своего фактического состояния; все, что он знает, — это лишь его доверительное состояние. Поэтому цикл принятия решений агентом POMDP состоит в следующем.

1. С учетом текущего доверительного состояния  $b$  выполнить действие  $a = \pi^*(b)$ .

---

<sup>3</sup> Модель наблюдения по сути идентична **модели восприятия** для временных процессов, как было описано в главе 15. Как и функция вознаграждения для задач MDP, модель наблюдения также может зависеть от действия и результирующего состояния, но связанное с этим изменение не является фундаментальным.

2. Получить результаты наблюдения  $o$ .
3. Установить текущее доверительное состояние равным  $\text{Forward}(b, a, o)$  и повторить ту же процедуру.

Теперь мы можем рассматривать задачи POMDP как требующие поиска в пространстве доверительных состояний, во многом аналогично тем методам, которые применялись для решения задач в отсутствие датчиков и задач в условиях непредвиденных ситуаций, описанных в главе 3. Основное различие состоит в том, что пространство доверительных состояний POMDP является непрерывным, поскольку доверительное состояние POMDP — это распределение вероятностей. Например, доверительное состояние для мира  $4 \times 3$  представляет собой точку в 11-мерном непрерывном пространстве. Любое действие изменяет не только физическое состояние, но и доверительное состояние, поэтому оценивается согласно информации, полученной агентом в качестве результата. Таким образом, задачи POMDP должны включать стоимость информации в качестве одного из компонентов задачи принятия решений (см. раздел 16.6).

Рассмотрим более внимательно результаты действий. В частности, рассчитаем вероятность того, что агент, находящийся в доверительном состоянии  $b$ , достигнет доверительного состояния  $b'$  после выполнения действия  $a$ . Итак, если известно действие и последующее наблюдение, то уравнение 17.11 должно обеспечить получение детерминированного обновления доверительного состояния, иными словами,  $b' = \text{Forward}(b, a, o)$ . Безусловно, результаты последующего наблюдения еще не известны, поэтому агент может перейти в одно из нескольких возможных доверительных состояний  $b'$ , в зависимости от того наблюдения, которое последует за данным действием. Вероятность получения результатов наблюдения  $o$ , при условии, что действие  $a$  было выполнено в доверительном состоянии  $b$ , определяется путем суммирования по всем фактическим состояниям  $s'$ , которых может достичь агент:

$$\begin{aligned} P(o | a, b) &= \sum_{s'} P(o | a, s', b) P(s' | a, b) \\ &= \sum_{s'} O(s', o) P(s' | a, b) \\ &= \sum_{s'} O(s', o) \sum_s T(s, a, s') b(s) \end{aligned}$$

Обозначим вероятность достижения состояния  $b'$  из  $b$ , если дано действие  $a$ , как  $\tau(b, a, b')$ . В таком случае справедливо следующее соотношение:

$$\begin{aligned} \tau(b, a, b') &= P(b' | a, b) = \sum_o P(b' | o, a, b) P(o | a, b) \\ &= \sum_o P(b' | o, a, b) \sum_{s'} O(s', o) \sum_s T(s, a, s') b(s) \quad (17.12) \end{aligned}$$

где  $P(b' | o, a, b)$  равно 1, если  $b' = \text{Forward}(b, a, o)$ , и 0 — в противном случае.

Уравнение 17.12 можно рассматривать как определение модели перехода для пространства доверительных состояний. Мы можем также определить функцию вознаграждения для доверительных состояний (т.е. ожидаемое вознаграждение, относящееся к фактическим состояниям, в которых может находиться агент) следующим образом:

$$\rho(b) = \sum_s b(s) R(s)$$

Итак, создается впечатление, что  $\tau(b, a, b')$  и  $\rho(b)$  совместно определяют наблюдаемую задачу MDP в пространстве доверительных состояний. Кроме того, можно показать, что оптимальная стратегия для этой задачи MDP,  $\pi^*(b)$ , является также оптимальной стратегией для оригинальной задачи POMDP. Другими словами, ~~решение любой задачи POMDP в пространстве физических состояний можно свести к решению задачи MDP в соответствующем пространстве доверительных состояний.~~ Этот факт, возможно, станет менее удивительным, если мы вспомним, что доверительное состояние по определению всегда является наблюдаемым для агента.

Но необходимо отметить следующее: хотя мы свели задачу POMDP к задаче MDP, полученная задача MDP имеет непрерывное (а иногда многомерное, с большим количеством размерностей) пространство состояний. Для решения подобных задач MDP нельзя непосредственно применить ни один из алгоритмов MDP, описанных в разделах 17.2 и 17.3. Но, как оказалось, существует возможность разработать версии алгоритмов итерации по значениям и итерации по стратегиям, которые могут применяться для решения задач MDP с непрерывными состояниями. Основная идея состоит в том, что стратегия  $\pi(b)$  может быть представлена как множество областей пространства доверительных состояний, каждая из которых связана с конкретным оптимальным действием<sup>4</sup>. Функция стоимости связывает с каждой из областей отдельную линейную функцию от  $b$ . На каждом этапе итерации по значениям или по стратегиям уточняются границы областей и могут вводиться новые области.

Подробное описание соответствующих алгоритмов выходит за рамки данной книги, но мы сообщаем решение для мира  $4 \times 3$  без датчиков. Оптимальная стратегия состоит в следующем:

[Left, Up, Up, Right, Up, Up, Right, Up, Up, Right, Up, Right, Up, Right, Up, ...]

Этот стратегия представляет собой последовательность, поскольку рассматриваемая задача в пространстве доверительных состояний является детерминированной — в ней нет наблюдений. Воплощенный в этой стратегии “секрет” состоит в том, что нужно предусмотреть для агента однократное движение *Left* для проверки того, что он не находится в квадрате  $(4, 1)$ , с тем чтобы в дальнейшем было достаточно безопасно продолжать движения *Up* и *Right* для достижения выхода +1. Агент достигает выхода +1 в 86,6% попыток и добивается этого гораздо быстрее по сравнению со стратегией, приведенной выше в данном разделе, поэтому его ожидаемая полезность повышается до 0.38, что гораздо больше по сравнению с 0.08.

---

<sup>4</sup> Для некоторых задач POMDP оптимальная стратегия имеет бесконечное количество областей, поэтому простой подход с использованием списка областей не позволяет добиться успеха, и для поиска даже приближенного решения нужны более изощренные методы.

Для более сложных задач POMDP с непустыми результатами наблюдений приближенный поиск оптимальных стратегий является очень сложным (фактически такие задачи являются PSPACE-трудными, т.е. действительно чрезвычайно трудными). Задачи с несколькими десятками состояний часто оказываются неразрешимыми. В следующем разделе описан другой, приближенный метод решения задач POMDP, основанный на опережающем поиске.

## 17.5. АГЕНТЫ, ДЕЙСТВУЮЩИЕ НА ОСНОВЕ ТЕОРИИ РЕШЕНИЙ

В этом разделе будет описан исчерпывающий подход к проектированию агентов для частично наблюдаемых, стохастических вариантов среды. Как показано ниже, основные элементы этого проекта должны быть уже знакомы читателю.

- Модели перехода и наблюдения представлены в виде **динамических байесовских сетей** (см. главу 15).
- Динамическая байесовская сеть дополняется узлами принятия решений и узлами полезности, по аналогии с теми, которые использовались в **сетях принятия решений** в главе 16. Результатирующая модель называется **динамической сетью принятия решений** (Dynamic Decision Network — DDN).
- Для учета данных о каждом новом восприятии и действии и для обновления представления доверительного состояния используется алгоритм фильтрации.
- Решения принимаются путем проектирования в прямом направлении возможных последовательностей действий и выбора наилучших из этих последовательностей.

Основное преимущество использования динамической байесовской сети для представления модели перехода и модели восприятия состоит в том, что такая сеть позволяет применять декомпозицию описания состояния на множество случайных переменных во многом аналогично тому, как в алгоритмах планирования используются логические представления для декомпозиции пространства состояний, применяемого в алгоритмах поиска. Поэтому проект агента представляет собой практическую реализацию **агента, действующего с учетом полезности**, который был кратко описан в главе 2.

Поскольку в этом разделе будут использоваться динамические байесовские сети, вернемся к системе обозначений главы 15, где символом  $\mathbf{x}_t$  обозначается множество переменных состояния во время  $t$ , а  $\mathbf{e}_t$  — переменные свидетельства. Таким образом, там, где до сих пор в этой главе использовалось обозначение  $s_t$  (состояние во время  $t$ ), теперь будет применяться обозначение  $\mathbf{x}_t$ . Для обозначения действия во время  $t$  будет использоваться запись  $A_t$ , поэтому модель перехода  $T(s, a, s')$  представляет собой не что иное, как  $P(\mathbf{x}_{t+1} | \mathbf{x}_t, A_t)$ , а модель наблюдения  $O(s, o)$  — то же, что и  $P(\mathbf{e}_t | \mathbf{x}_t)$ . Для обозначения вознаграждения, полученного во время  $t$ , будет применяться запись  $R_t$ , а для обозначения полезности состояния во время  $t$  — запись  $U_t$ . При использовании такой системы обозначений динамическая сеть принятия решений принимает вид, подобный показанному на рис. 17.7.

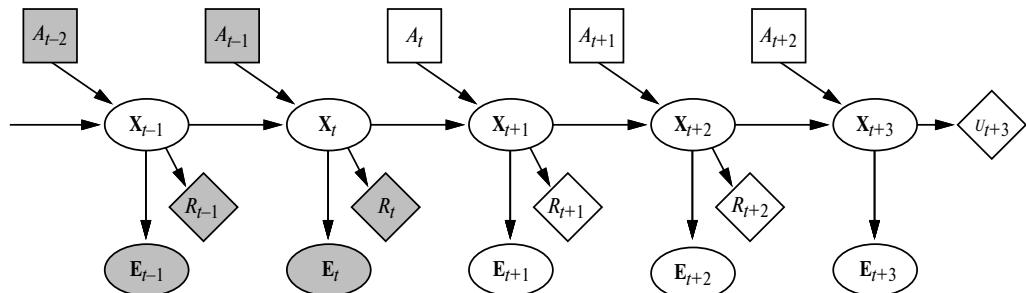


Рис. 17.7. Универсальная структура динамической сети принятия решений. Переменные с известными значениями выделены затенением. Текущим временем является  $t$ , а агент должен решить, что делать дальше, т.е. выбрать значение для  $A_t$ . Сеть развернута в будущее на три этапа и представляет будущее вознаграждение, а также полезность состояния на горизонте прогноза

Динамические сети принятия решений позволяют получить краткое представление крупных задач POMDP, поэтому могут использоваться в качестве входных данных для любого алгоритма POMDP, включая те из них, которые относятся к методам итерации по значениям и итерации по стратегиям. В этом разделе мы сосредоточимся на прогностических методах, которые проектируют последовательности действий в прямом направлении от текущего доверительного состояния во многом таким же образом, как и алгоритмы ведения игр, описанные в главе 6. Сеть, приведенная на рис. 17.7, спроектирована в будущее на три этапа; неизвестными являются все текущие и будущие решения, а также будущие наблюдения и вознаграждения. Обратите внимание на то, что сеть включает узлы вознаграждений для  $X_{t+1}$  и  $X_{t+2}$ , но для  $X_{t+3}$  — только узел полезности. Это связано с тем, что агент должен максимизировать (обесцениваемую) сумму всех будущих вознаграждений, а полезность  $U(X_{t+3})$  представляет и вознаграждение, относящееся к  $X_{t+3}$ , и все будущие вознаграждения. Как и в главе 6, предполагается, что значение полезности  $U$  доступно только в некоторой приближенной форме, поскольку, если бы были известны точные значения полезности, то не было бы необходимости заглядывать вперед на глубину больше 1.

На рис. 17.8 показана часть дерева поиска, соответствующего приведенной на рис. 17.7 сети DDN, развернутой в будущее на три этапа. Каждый из узлов, обозначенных треугольником, представляет собой доверительное состояние, в котором агент принимает решение  $A_{t+i}$  для  $i=0, 1, 2, \dots$ . Узлы, обозначенные кружками, соответствуют выборам, сделанным средой, а именно тому, какие получены результаты наблюдений  $E_{t+i}$ . Обратите внимание на то, что в этой сети нет узлов жеребьевки, соответствующих результатам действий; это связано с тем, что обновление доверительного состояния для любого действия является детерминированным, независимо от фактического результата.

Доверительное состояние каждого обозначенного треугольником узла можно вычислить, применив алгоритм фильтрации к ведущей к нему последовательности наблюдений и действий. Таким образом, в алгоритме учитывается тот факт, что для принятия решения о выполнении действия  $A_{t+i}$  агент должен иметь доступные результаты восприятия  $E_{t+1}, \dots, E_{t+i}$ , даже несмотря на то, что во время  $t$  он не знает, какими будут эти результаты восприятия. Благодаря этому агент, действующий на основе теории принятия решений, автоматически учитывает стоимость информации и выполняет действия по сбору информации, когда это потребуется.

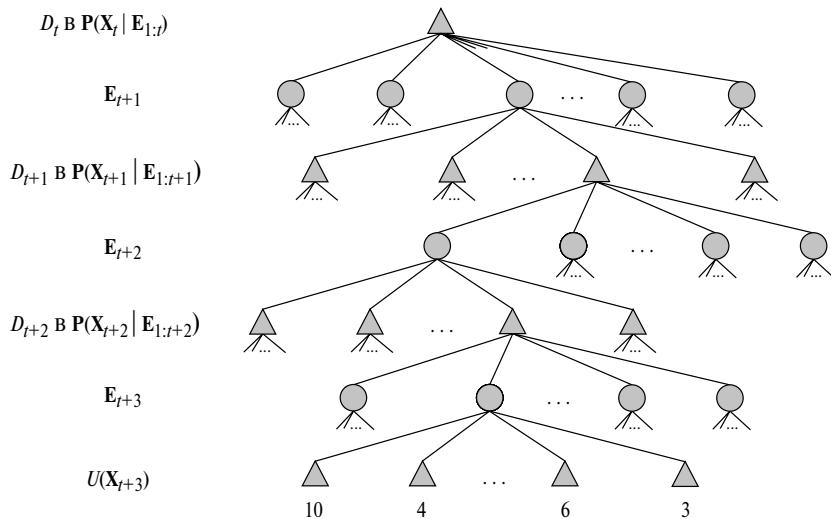


Рис. 17.8. Часть прогнозистического решения для сети DDN, приведенной на рис. 17.7

Из данного дерева поиска можно извлечь информацию о решении, резервируя значения полезности, взятые из листовых узлов, вычисляя среднее в узлах жеребьевки и определяя максимальные значения в узлах принятия решений. Такая организация работы аналогична применяемой в алгоритме Expectiminimax для деревьев игр с узлами жеребьевки, за исключением того, что, во-первых, вознаграждения могут быть предусмотрены и в состояниях, отличных от листовых, и, во-вторых, узлы принятия решений соответствуют доверительным состояниям, а не фактическим состояниям. Временная сложность исчерпывающего поиска до глубины  $d$  выражается как  $O(|D|^d \cdot |\mathbf{E}|^d)$ , где  $|D|$  — количество доступных действий;  $\mathbf{E}$  — количество возможных наблюдений. Для получения решений, близких к оптимальным, при решении таких задач, в которых коэффициент обесценивания  $\gamma$  не слишком близок к 1, часто бывает вполне приемлемым поверхностный поиск. Существует также возможность аппроксимировать этап усреднения в узлах жеребьевки, осуществляя выборку из множества возможных наблюдений вместо суммирования по всем возможным наблюдениям. Могут также применяться некоторые другие способы быстрого поиска качественных приближенных решений, но мы отложим их описание до главы 21.

Действующие по принципам теории решений агенты, основанные на динамических сетях принятия решений, имеют целый ряд преимуществ по сравнению с другими, более простыми агентами, проекты которых представлены в предыдущих главах. В частности, они способны функционировать в частично наблюдаемых неопределенных вариантах среды и могут легко пересматривать свои “планы” с учетом непредвиденных результатов наблюдений. При использовании подходящих моделей восприятия эти агенты могут справиться с ситуациями наподобие отказа датчиков и способны составлять планы по сбору информации. Под давлением жестких требований ко времени и в сложных вариантах среды эти агенты проявляют способность осуществлять “корректный переход к более примитивному поведению” (graceful degradation) с использованием различных методов вычисления приближенных решений. Так чего же в них недостает? Наиболее важным недостатком агентов, осно-

ванных на использовании алгоритмов динамической сети принятия решений, является то, что в них используется прямой поиск, полностью аналогичный тому, который предусмотрен в алгоритмах поиска в пространстве состояний, описанных в части II. В части IV было показано, что способность рассматривать частично упорядоченные, абстрактные планы с использованием целенаправленного поиска обеспечивает существенное расширение возможностей решения задач, особенно если при этом применяются библиотеки планов. Кроме того, были предприняты попытки распространить эти методы на вероятностную проблемную область, но до сих пор они оказывались неэффективными. Еще одной связанной с этим проблемой является слишком простой язык динамических сетей принятия решений, который по сути является пропозициональным. Было бы желательно распространить на задачи принятия решений некоторые идеи вероятностных языков первого порядка, описанные в разделе 14.6. Современные исследования показали, что такое расширение возможно и что оно позволяет получить существенные преимущества, как описано в заметках в конце данной главы.

## 17.6. ПРИНЯТИЕ РЕШЕНИЙ ПРИ НАЛИЧИИ НЕСКОЛЬКИХ АГЕНТОВ: ТЕОРИЯ ИГР

Данная глава в основном посвящена теме принятия решений в неопределенных вариантах среды. А как обстоят дела в том случае, если неопределенность связана с наличием других агентов и осуществлением ими решений, которые они принимают? И что будет, если на решения этих агентов, в свою очередь, влияют решения нашего агента? Эти вопросы уже рассматривались в настоящей книге при описании игр в главе 6. Но в этой главе речь в основном шла об играх с полной информацией, в которых игроки делают ходы по очереди: в подобных играх для определения оптимальных ходов может использоваться минимаксный поиск. А в данном разделе рассматриваются некоторые идеи теории игр, которые могут применяться при анализе игр с одновременно выполняемыми ходами. Для упрощения изложения вначале рассмотрим игры, которые продолжаются только в течение одного хода. На первый взгляд может показаться, что слово “игра” не совсем подходит для обозначения такого упрощения, которое сводится к одному ходу, но в действительности теория игр используется в очень серьезных ситуациях принятия решений, включая ведение дел о банкротстве, организацию аукционов по распределению спектра радиочастот, принятие решений по разработке промышленной продукции и назначению на нее цен, а также национальную оборону. В таких ситуациях речь часто идет о миллиардах долларов и сотнях тысяч человеческих жизней. Теория игр может использоваться по меньшей мере в двух описанных ниже направлениях.

1. **Проектирование агента.** Теория игр позволяет анализировать решения агента и вычислять ожидаемую полезность для каждого решения (с учетом предположения, что другие агенты действуют оптимальным образом согласно теории игр). Например, в игре **в чет и нечет на двух пальцах** (этую игру на пальцах называют также *morra*, от итальянского слова *cattorra* — группа) два игрока, *O* (Odd — нечетный) и *E* (Even — четный), одновременно показывают один или два пальца. Допустим, что общее количество показанных пальцев равно *f*. Если число *f* является нечетным, игрок *O* получает *f* долларов от игрока *E*,

а если число  $f$  — четное, игрок  $E$  получает  $f$  долларов от игрока  $O$ . Теория игр позволяет определить наилучшую стратегию в игре против рационально действующего игрока и ожидаемый выигрыш для каждого игрока<sup>5</sup>.

- 2. Проектирование механизма.** Если в среде присутствует много агентов, то может существовать возможность так определить правила действий в этой среде (т.е. правила игры, в которую должны играть агенты), чтобы общее благосостояние всех агентов максимизировалось, если каждый агент принимает обоснованное теорией игр решение, максимизирующее его собственную полезность. Например, теория игр позволяет проектировать такие протоколы для коллекции маршрутизаторов трафика Internet, чтобы каждый маршрутизатор стремился действовать в направлении максимизации глобальной пропускной способности. Проектирование механизма может также использоваться для создания интеллектуальных **мультиагентных систем**, которые решают сложные задачи в распределенной форме без необходимости для каждого агента знать о том, какая задача решается в целом.

Любая игра в теории игр определяется с помощью описанных ниже компонентов.

- **Игроки**, или агенты, которые должны принимать решения. Наибольшее внимание в исследованиях уделялось играм с двумя игроками, хотя достаточно часто рассматриваются также игры с  $n$  игроками, где  $n > 2$ . В этой главе имена игроков записываются с прописной буквы, например *Alice* и *Bob* или *O* и *E*.
- **Действия**, которые могут быть выбраны игроками. В этой главе имена действий записываются строчными буквами, например *one* или *testify*. В распоряжении игроков могут находиться одинаковые или неодинаковые множества действий.
- **Матрица вознаграждений**, которая определяет полезность для каждого игрока каждой комбинации действий всех игроков. Матрица вознаграждений для игры чет или нечет на двух пальцах приведена в табл. 17.1 (*one* — выбор действия, в котором игрок показывает один палец, *two* — два пальца).

Таблица 17.1. Матрица вознаграждений для игры чет или нечет на двух пальцах

	<i>O: one</i>	<i>O: two</i>
<i>E: one</i>	$E=2, O=-2$	$E=-3, O=3$
<i>E: two</i>	$E=-3, O=3$	$E=4, O=-4$

Например, в нижнем правом углу показано, что если игрок  $O$  выбирает действие *two*, а игрок  $E$  также выбирает *two*, то вознаграждение для  $E$  равно 4, а для  $O$  равно -4.

Каждый игрок, участвующий в игре, должен выработать, а затем осуществить **стратегию** (напомним, что такой же термин используется для обозначения способа выбора действий не только в теории игр, но и в теории принятия решений). **Чистая стратегия** — это детерминированная стратегия, определяющая конкрет-

<sup>5</sup> Игра в чет и нечет — это развлекательная версия реальной **игры в инспекцию**. В таких играх инспектор выбирает день для инспектирования предприятия (такого как ресторан или завод биологического оружия), а оператор предприятия выбирает день, в который нужно спрятать все запрещенные предметы. Если эти дни не совпадают, выигрывает инспектор, а если совпадают — выигрывает оператор предприятия.

ное действие, которое должно быть выполнено в каждой ситуации; в игре, состоящей из одного хода, чистая стратегия состоит из одного действия. Анализ игр приводит к формулировке идеи смешанной стратегии, которая представляет собой рандомизированную стратегию, предусматривающую выбор конкретных действий среди доступных действий в соответствии с определенным распределением вероятностей. Смешанная стратегия, в которой обычно выбирается действие  $a$  с вероятностью  $p$ , а в противном случае выбирается действие  $b$ , условно обозначается как  $[p : a; (1-p) : b]$ . Например, смешанной стратегией для игры в чет и нечет на двух пальцах может быть  $[0.5 : \text{one}; 0.5 : \text{two}]$ . Профилем стратегии называется вариант присваивания стратегии каждому игроку; после того как задан профиль стратегии, результат игры для каждого игрока принимает определенное числовое значение.

Решением игры называется профиль стратегии, в котором каждый игрок принимает рациональную стратегию. Как будет показано ниже, наиболее важной проблемой в теории игр является определение того, что подразумевается под словом “рациональный”, когда каждый агент выбирает только часть профиля стратегии, определяющую этот результат. Важно понять, что результаты представляют собой фактические итоги ведения игры, а решения являются теоретическими конструкциями, которые используются для анализа игры. Ниже будет показано, что некоторые игры имеют решение только в смешанных стратегиях. Но это не означает, что игрок должен в буквальном смысле слова принимать смешанную стратегию, чтобы действовать рационально.

Рассмотрим описанную ниже историю. Два отпетых грабителя, Алиса и Боб, были пойманы с поличным недалеко от места совершенного ими ограбления, и теперь их отдельно допрашивают следователи. Оба они знают, что, признавшись в совместном совершении этого преступления, получат по 5 лет тюремного заключения за грабеж, а если оба откажутся признаваться, то получат только по 1 году каждый за менее грубое правонарушение — владение краденым имуществом. Но следователи предлагают отдельно каждому из них такую сделку: если вы дадите свидетельство против вашего партнера как главаря шайки грабителей, то вас освободят, а партнера осудят на 10 лет. Теперь Алиса и Боб сталкиваются с так называемой дилеммой заключенного: должны ли они свидетельствовать (*testify*) или отказаться (*refuse*)? Будучи рациональными агентами, и Алиса, и Боб желают максимизировать свою собственную ожидаемую полезность. Предположим, что Алиса полностью безразлична к судьбе своего партнера, а ее оценка полезности уменьшается пропорционально количеству лет, которые она проведет в тюрьме, независимо от того, что случится с Бобом. Боб думает точно так же. Чтобы упростить для себя выработку рационального решения, оба грабителя составляют матрицу вознаграждений, приведенную в табл. 17.2.

Таблица 17.2. Матрица вознаграждений для дилеммы заключенного

	<i>Alice: testify</i>	<i>Alice: refuse</i>
<i>Bob: testify</i>	$A = -5, B = -5$	$A = -10, B = 0$
<i>Bob: refuse</i>	$A = 0, B = -10$	$A = -1, B = -1$

Алиса анализирует эту матрицу вознаграждений следующим образом: предположим, что Боб будет свидетельствовать против меня. В таком случае я получу 5 лет,

если буду свидетельствовать против него, и 10 лет — если откажусь, поэтому в данном случае лучше свидетельствовать против него. С другой стороны, если Боб откажется, то я получу 0 лет, если буду свидетельствовать, и 1 год, если откажусь, поэтому и в данном случае лучше свидетельствовать. Итак, и в том и в другом случае для меня лучше свидетельствовать против Боба, поэтому так я и должна поступить.

Алиса обнаружила, что *testify* — это **dominantная стратегия** для рассматриваемой игры. Принято считать, что стратегия  $s$  для игрока  $p$  **строго доминирует** над стратегией  $s'$ , если результат стратегии  $s$  лучше для игрока  $p$ , чем результат стратегии  $s'$ , при любом выборе стратегий другими игроками. Стратегия  $s$  **слабо доминирует** над  $s'$ , если  $s$  лучше чем  $s'$  по меньшей мере в одном профиле стратегий и не хуже в любом другом. *Доминантной стратегией* является стратегия, которая доминирует над всеми остальными. Нерационально вести игру на основе стратегии, над которой строго доминируют другие стратегии, и нерационально не вести игру на основе доминантной стратегии, если она существует. Будучи рациональным агентом, Алиса выбирает доминантную стратегию. Прежде чем перейти к дальнейшему изложению, необходимо рассмотреть еще несколько терминов: принято считать, что некоторый результат является **оптимальным согласно принципу Парето**<sup>6</sup>, если нет другого результата, который предпочли бы все игроки. Над результатом **доминирует по принципу Парето** другой результат, если все игроки предпочитают этот другой результат.

А если Алиса — не только рациональный, но и умный агент, она должна продолжить рассуждение следующим образом: доминантной стратегией Боба также является свидетельствование против меня. Поэтому он будет свидетельствовать, и мы оба получим по пять лет. Если оба игрока имеют доминантную стратегию, то комбинация этих стратегий называется **равновесием доминантных стратегий**. Вообще говоря, профиль стратегий образует **равновесие**, если ни один игрок не может извлечь выгоду от переключения с одной стратегии на другую, при условии, что все остальные игроки продолжают придерживаться одной и той же стратегии. Равновесие по сути представляет собой **локальный оптимум** в пространстве стратегий; оно является вершиной пика, вокруг которого находятся спады вдоль всех измерений, где измерения соответствуют вариантам выбора стратегии игроком.

Парадоксальность дилеммы заключенного состоит в том, что результат в точке равновесия для обоих игроков хуже по сравнению с результатом, которого они достигли бы, если бы оба отказались свидетельствовать друг против друга. Иными словами, результат для равновесного решения доминируется по принципу Парето результатом  $(-1, -1)$ , который соответствует обоюдному отказу от свидетельства, (*refuse, refuse*).

Существует ли какой-либо способ, с помощью которого Алиса и Боб могли бы формально прийти к результату  $(-1, -1)$ ? Безусловно, что для них обоих вариант, в котором они отказываются свидетельствовать, является допустимым, но этот вариант маловероятен. Любой игрок, анализирующий вариант хода *refuse*, поймет, что для него было бы лучше выбрать ход *testify*. В этом состоит притягательная мощь точки равновесия.

Математик Джон Нэш (род. в 1928 году) доказал теорему, согласно которой **каждая игра имеет равновесие такого типа, которое определено в данном примере**.

---

<sup>6</sup> Оптимальность по принципу Парето названа в честь экономиста Вильфредо Парето (1848–1923).

Теперь указанное условие называют в его честь **равновесием Нэша**. Очевидно, что равновесием домinantных стратегий является равновесие Нэша (упр. 17.9), но не все игры имеют домinantные стратегии. Теорема Нэша означает, что равновесные стратегии могут существовать даже при отсутствии домinantных стратегий.

В дилемме заключенного равновесием Нэша является только профиль стратегий (*testify, testify*). Трудно понять, как рациональные игроки могут избежать такого результата, поскольку в любом предложенном неравновесном решении по меньшей мере один из игроков будет подвергаться соблазну изменить свою стратегию. Специалисты в области теории игр соглашаются с тем, что обязательным условием того, чтобы некоторый ход представлял собой решение, является принадлежность его к равновесию Нэша, но еще не достигнутое полное согласие в отношении того, является ли это условие достаточным.

Решения (*testify, testify*) можно довольно легко избежать, немного изменив правила игры (или взгляды игроков). Например, можно перейти к итерационной игре, в которой игроки знают, что когда-то обязательно снова встретятся и вспомнят, как действовали при прошлой встрече (но крайне важно то, что они не должны иметь определенной информации о том, через какое время встретятся). Кроме того, если агенты имеют моральные убеждения, которые способствуют сотрудничеству и справедливому отношению друг к другу, то можно изменить матрицу вознаграждения так, чтобы она отражала для каждого агента полезность взаимодействия с другим агентом. Как будет показано ниже, на результатах может также отразиться такая модификация агентов, чтобы они обладали ограниченной вычислительной мощью, а не способностью рассуждать абсолютно рационально, а также предоставление одному агенту информации о том, что способности другого рассуждать рационально ограничены.

Теперь рассмотрим игру, в которой отсутствует домinantная стратегия. Предположим, что компания Acme, изготовитель аппаратных средств для видеоигр, должна решить, будут ли в ее следующей игровой машине использоваться DVD или CD. Между тем компания Best, производитель программного обеспечения для видеоигр, должна принять решение о том, следует ли выпускать свою очередную игру на DVD или CD. Для обеих компаний прибыль будет положительной, если они примут согласованные решения, и отрицательной, если решения не будут согласованными, как показывает матрица вознаграждений, приведенная в табл. 17.3.

**Таблица 17.3. Матрица вознаграждений для компаний Acme и Best**

	<i>Acme: dvd</i>	<i>Acme: cd</i>
<i>Best: dvd</i>	$A=9, B=9$	$A=-4, B=-1$
<i>Best: cd</i>	$A=-3, B=-1$	$A=5, B=5$

Для этой игры отсутствует равновесие домinantных стратегий, но имеются два равновесия Нэша: (*dvd, dvd*) и (*cd, cd*). Известно, что это — равновесия Нэша, поскольку, если один из игроков примет одностороннее решение перейти на другую стратегию, ситуация для него ухудшится. Теперь эти агенты сталкиваются с такой проблемой: *имеется несколько приемлемых решений, но если каждый агент выберет другое решение, то результирующий профиль стратегий вообще не будет представлять собой решение, и оба агента понесут убытки.* Каким образом эти игроки могут согласованно прийти к какому-то решению? Один из возможных ответов состоит в

том, что оба игрока должны выбрать решение, оптимальное по принципу Парето, —  $(dvd, dvd)$ ; это означает, что можно ограничить определение понятия “решения” уникальным, оптимальным по принципу Парето равновесием Нэша, при условии, что оно существует. Каждая игра имеет по меньшей мере одно оптимальное по принципу Парето решение, но в любой игре может быть несколько таких решений, или эти решения могут не оказаться точками равновесия. Например, можно было бы установить вознаграждения за решение  $(dvd, dvd)$ , равные 5, а не 9. В этом случае имеются две одинаковые точки равновесия, оптимальные по принципу Парето. Чтобы выбрать между ними, агенты должны либо пользоваться догадками, либо прибегать к общению, что можно сделать, приняв соглашение об упорядочении решений до начала игры, или проводить переговоры, чтобы достичь обоюдно выгодного решения во время игры (это может означать, что в состав многоходовой игры должны быть включены коммуникативные действия). Таким образом, необходимость в общении возникает в рамках теории игр точно по тем же причинам, что и в мультиагентном планировании, описанном в главе 12. Игры, подобные этой, в которых игроки должны вступать во взаимодействие, называются **координационными играми**.

Выше было показано, что игра может иметь больше одного равновесия Нэша; но на основании чего мы можем утверждать, что каждая игра должна иметь по меньшей мере одно такое равновесие? Может оказаться, что в игре отсутствует равновесие Нэша для чистой (не смешанной) стратегии. Рассмотрим, например, любой профиль чистых стратегий для игры в чет и нечет на двух пальцах (с. 839). Если общее количество показанных пальцев является четным, то игрок  $O$  может захотеть переключиться на другую стратегию, а если это количество нечетно, то будет стремиться переключиться игрок  $E$ . Поэтому ни один профиль чистых стратегий не может представлять собой равновесие и приходится рассматривать смешанные стратегии.

Но какой должна быть смешанная стратегия? В 1928 году фон Нейман разработал метод поиска оптимальной смешанной стратегии для игр с двумя игроками, называемых **играми с нулевой суммой**. Играй с нулевой суммой называется игра, в которой вознаграждения в каждой ячейке матрицы вознаграждения в сумме равны нулю<sup>7</sup>. Очевидно, что игра в чет и нечет является именно таковой. Известно, что для игр с двумя игроками и нулевой суммой вознаграждения являются равными и противоположными, поэтому достаточно рассмотреть вознаграждения только для одного игрока, который будет считаться максимизирующим игроком (точно так же, как и в главе 6). Применительно к игре в чет и нечет выберем в качестве максимизирующего игрока  $E$ , который выбрал для себя в качестве выигрышного результата четное количество пальцев, поэтому можно определить матрицу вознаграждений на основе значений  $U_E(e, o)$  — вознаграждение, получаемое игроком  $E$ , если игрок  $E$  выбирает действие  $e$ , а  $O$  — действие  $o$ .

Метод фон Неймана называется методом **максимина** и действует, как описано ниже.

<sup>7</sup> Более общим является понятие **игр с постоянной суммой**, в которых сумма в каждой ячейке матрицы вознаграждений игры равна некоторой константе  $c$ . Любую игру с постоянной суммой и  $n$  участниками можно преобразовать в игру с нулевой суммой, вычитя из каждого вознаграждения значение  $c/n$ . Таким образом, шахматы, в которых по традиции применяется вознаграждение 1 за победу,  $1/2$  — за ничью и 0 — за проигрыш, формально представляет собой игру с постоянной суммой, где  $c=1$ , но ее можно легко преобразовать в игру с нулевой суммой, вычитя  $1/2$  из каждого вознаграждения.

- Предположим, что правила игры изменились таким образом, что игрок  $E$  вынужден раскрывать свою стратегию первым, а за ним следует игрок  $O$ . В таком случае мы имеем дело с игрой, где ходы выполняются поочередно, к которой можно применить стандартный **минимаксный** алгоритм из главы 6. Допустим, что такая игра приводит к получению результата  $U_{E,O}$ . Очевидно, что эта игра окончится в пользу игрока  $O$ , поэтому истинная полезность  $U$  данной игры (с точки зрения игрока  $E$ ) равна по меньшей мере  $U_{E,O}$ . Например, если рассматриваются только чистые стратегии, то минимаксное дерево игры имеет корневое значение, равное  $-3$  (рис. 17.9,  $a$ ), поэтому известно, что  $U \geq -3$ .
- Теперь предположим, что правила изменились таким образом, что свою стратегию вынужден раскрывать игрок  $O$ , а за ним следует игрок  $E$ . В таком случае минимаксное значение этой игры становится равным  $U_{O,E}$ , а поскольку игра складывается в пользу игрока  $E$ , то известно, что полезность  $U$  самое большое равна  $U_{O,E}$ . При использовании чистых стратегий это значение равно  $+2$  (см. рис. 17.9,  $b$ ), поэтому известно, что  $U \leq +2$ .

Рассматривая эти два предположения совместно, можно прийти к заключению, что истинная полезность  $U$  рассматриваемого решения должна удовлетворять следующему неравенству:

$$U_{E,O} \leq U \leq U_{O,E} \text{ или в данном случае } -3 \leq U \leq 2$$

Чтобы точно определить значение  $U$ , необходимо перейти к анализу смешанных стратегий. Вначале отметим следующее:  *как только первый игрок раскрыл свою стратегию, второй игрок не может проиграть, ведя игру согласно чистой стратегии.* Причина этого проста — если второй игрок ведет игру на основе смешанной стратегии,  $[p : one; (1-p) : two]$ , то ожидаемая полезность этой игры представляет собой линейную комбинацию  $(p \cdot u_{one} + (1-p) \cdot u_{two})$  полезностей чистых стратегий  $u_{one}$  и  $u_{two}$ . Эта линейная комбинация ни при каких условиях не будет лучше по сравнению с лучшим из значений  $u_{one}$  и  $u_{two}$ , поэтому второй игрок вполне может просто выбрать для ведения игры чистую стратегию.

С учетом этого замечания минимаксные деревья можно рассматривать как имеющие бесконечное количество ветвей, исходящих от корня, которые соответствуют бесконечному количеству смешанных стратегий, доступных для выбора первым игроком. Каждая из этих ветвей ведет к узлу с двумя ветвями, соответствующими чистым стратегиям для второго игрока. Эти бесконечные деревья можно изобразить как конечные, предусмотрев один “параметризованный” выбор у корня, как описано ниже.

- Ситуация, возникающая, если игрок  $E$  ходит первым, показана на рис. 17.9,  $a$ . Игрок  $E$  делает из корневой позиции ход  $[p : one; (1-p) : two]$ , а затем игрок  $O$  выбирает ход с учетом значения  $p$ . Если игрок  $O$  выбирает ход *one*, то ожидаемое вознаграждение (для  $E$ ) становится равным  $2p - 3(1-p) = 5p - 3$ ; если игрок  $O$  выбирает ход *two*, то ожидаемое вознаграждение равно  $-3p + 4(1-p) = 4 - 7p$ . Зависимости, выраждающие величину этих двух вознаграждений, можно изобразить в виде прямых линий на графике, где  $p$  изменяется от 0 до 1 вдоль оси  $x$ , как показано на рис. 17.9,  $d$ . Игрок  $O$ , минимизирующий стоимость игры, должен всегда выбирать наименьшее значение на двух прямых линиях, как показано на этом рисунке жирными отрезками прямых. Поэтому наилучшее решение, которое может

принять игрок  $E$ , выбирая ход, подлежащий выполнению из корневой позиции, состоит в том, чтобы выбрать значение  $p$ , соответствующее точке пересечения и определяемое следующим образом:

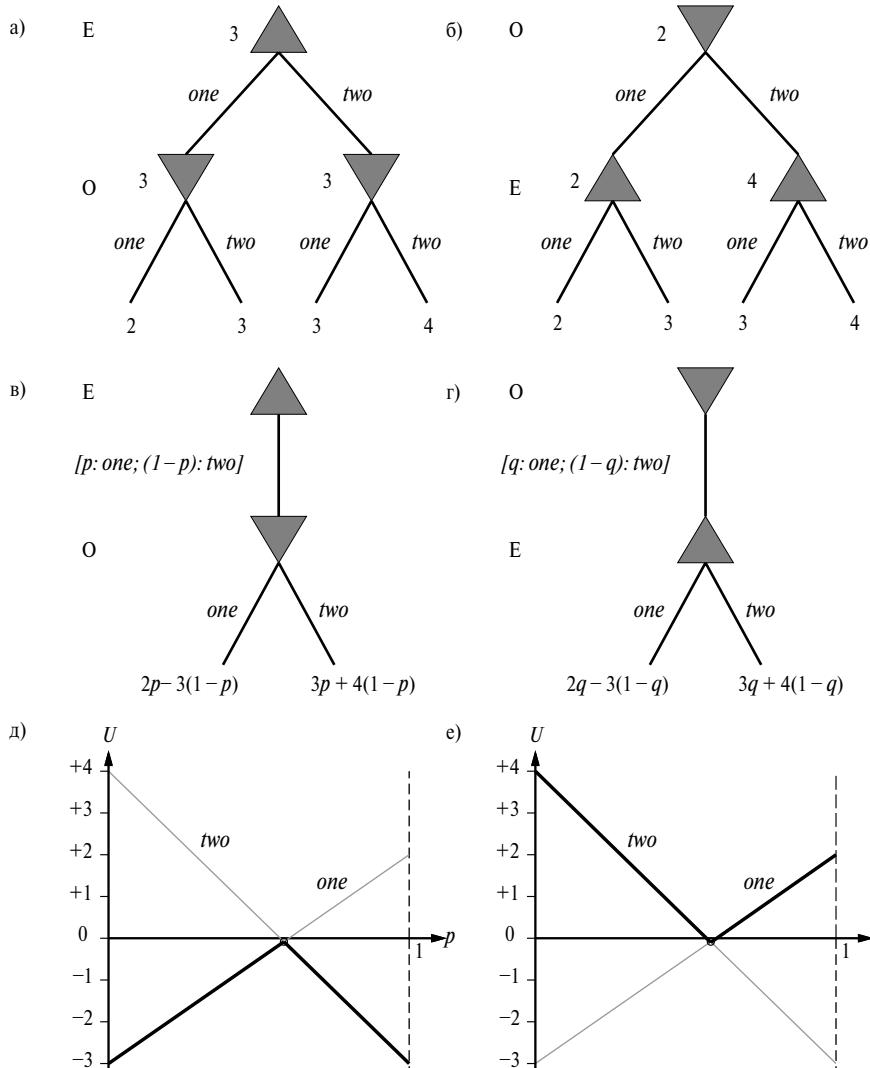


Рис. 17.9. Анализ игры с двумя игроками: минимаксные деревья игры в чет и нечет на двух пальцах, если игроки ходят по очереди, ведя игру на основе чистых стратегий (а), (б); параметризованные деревья игры, в которой первый игрок использует смешанную стратегию, причем вознаграждения зависят от параметра вероятности ( $p$  или  $q$ ) в смешанной стратегии (с), (д); для любого конкретного значения параметра вероятности второй игрок выбирает “наилучшее” этих двух действий, поэтому значение для смешанной стратегии первого игрока задается жирными линиями; первый игрок выбирает параметр вероятности для смешанной стратегии в точке пересечения (д), (е)

$$5p - 3 = 4 - 7p \Rightarrow p = 7/12$$

Полезность для игрока  $E$  в этот момент времени равна  $U_{E,0} = -1/12$ .

- А если первым ходит игрок  $O$ , то складывается ситуация, показанная на рис. 17.9, г. Игрок  $O$  из корневой позиции делает ход  $[q : one; (1-q) : two]$ , после чего игрок  $E$  выбирает ход с учетом значения  $q$ . При этом вознаграждения определяются соотношениями<sup>8</sup>  $2q - 3(1-q) = 5q - 3$  и  $-3q + 4(1-q) = 4 - 7q$ . Опять-таки на рис. 17.9, е показано, что наилучший ход, который может быть сделан игроком  $O$  из корневой позиции, состоит в выборе точки пересечения, следующим образом:

$$5q - 3 = 4 - 7q \Rightarrow q = 7/12$$

Полезность для игрока  $E$  в этот момент равна  $U_{O,E} = -1/12$ .

Теперь известно, что истинная полезность этой игры находится в пределах от  $-1/12$  до  $-1/12$ , т.е. она точно равна  $-1/12$ ! (Общий вывод состоит в том, что в эту игру лучше играть от имени игрока  $O$ , а не  $E$ .) Кроме того, истинная полезность достигается при использовании смешанной стратегии  $[7/12 : one; 5/12 : two]$ , которой должны придерживаться оба игрока. Такая стратегия называется **максиминным равновесием** игры и соответствует равновесию Нэша. Обратите внимание на то, что каждая составляющая стратегия в равновесной смешанной стратегии имеет одну и ту же ожидаемую полезность. В данном случае и ход  $one$ , и ход  $two$  имеют ту же ожидаемую полезность,  $-1/12$ , что и сама смешанная стратегия.

Приведенный выше результат для игры в чет и нечет на двух пальцах представляет собой пример общего результата, полученного фон Нейманом: *каждая игра с нулевой суммой с двумя игроками имеет максиминное равновесие, если разрешены смешанные стратегии*. Кроме того, каждое равновесие Нэша в игре с нулевой суммой представляет собой максиминное равновесие для обоих игроков. Но общий алгоритм поиска максиминных равновесий в играх с нулевой суммой немного сложнее по сравнению с тем, что показано в виде схем на рис. 17.9, д и е. Если количество возможных действий равно  $n$ , то смешанная стратегия представляет собой точку в  $n$ -мерном пространстве и прямые линии становятся гиперплоскостями. Возможно также, что над некоторыми чистыми стратегиями для второго игрока будут доминировать другие стратегии, так что они перестанут быть оптимальными по отношению к любой стратегии для первого игрока. После удаления всех подобных стратегий (а эту операцию может потребоваться выполнить неоднократно) оптимальным вариантом хода из корневой позиции становится самая высокая (или самая низкая) точка пересечения оставшихся гиперплоскостей. Поиск этого варианта представляет собой пример задачи **линейного программирования** — максимизации целевой функции с учетом линейных ограничений. Такие задачи могут быть решены с помощью стандартных методов за время, полиномиально зависящее от количества действий (а также формально и от количества битов, используемых для определения функции вознаграждения).

---

<sup>8</sup> То, что эти уравнения являются такими же, как и для  $p$ , обусловлено лишь совпадением; совпадение возникло, поскольку  $U_E(one, two) = U_E(two, one) = -3$ . Это совпадение также объясняет, почему оптимальная стратегия является одинаковой для обоих игроков.

Остается нерешенным следующий вопрос: как фактически должен действовать рациональный агент при ведении такой одноходовой игры, как игра в чет и нечет? Рациональный агент пришел логическим путем к выводу, что  $[7/12 : one; 5/12 : two]$  представляет собой максиминную равновесную стратегию, и исходит из предположения, что знаниями об этом обладает и его рациональный противник. Агент может использовать игральную кость с 12 сторонами или генератор случайных чисел для выбора случайным образом хода, соответствующего этой смешанной стратегии, и в этом случае ожидаемое вознаграждение для игрока  $E$  будет равно  $-1/12$ . В ином случае агент может просто решить сделать ход  $one$  или  $two$ . В любом случае для игрока  $E$  ожидаемое вознаграждение остается равным  $-1/12$ . Удивительно то, что односторонний выбор конкретного действия не уменьшает ожидаемое вознаграждение для данного игрока, но если другой агент будет иметь возможность узнать, что данный игрок принял такое одностороннее решение, то ожидаемое вознаграждение изменится, поскольку противник сможет откорректировать свою стратегию соответствующим образом.

Поиск решений для конечных игр с ненулевой суммой (т.е. равновесий Нэша) немного сложнее. Общий подход заключается в использовании двух этапов. Во-первых, необходимо составить список из всех возможных последовательностей действий, которые могут образовывать смешанные стратегии. Например, вначале следует проверить все профили стратегий, в которых каждый игрок выполняет одно действие, затем те, в которых каждый игрок выполняет либо одно, либо два действия, и т.д. Количество таких проверок экспоненциально зависит от количества действий, поэтому может применяться только в относительно простых играх. Во-вторых, для каждого профиля стратегий, включенного в список на первом этапе, необходимо провести проверку для определения того, представляет ли он некоторое равновесие. Такая задача выполняется путем решения ряда уравнений и неравенств, аналогичных используемых в случае с нулевой суммой. В игре с двумя игроками эти уравнения являются линейными и могут быть решены с помощью основных методов линейного программирования, но в случае трех или большего количества игроков они являются нелинейными и задача поиска их решения может оказаться очень сложной.

До сих пор мы рассматривали только такие игры, которые состоят из одного хода. Простейшей разновидностью игры, состоящей из нескольких ходов, является **повторяющаяся игра**, в которой игроки снова и снова сталкиваются с одним и тем же выбором, но каждый раз пользуются знаниями истории всех предыдущих выборов всех игроков. Профиль стратегий для повторяющейся игры определяет выбор действия для каждого игрока на каждом временном интервале для всех возможных историй предыдущих выборов. Как и в случае задач MDP, вознаграждения определяются аддитивной функцией от времени.

Рассмотрим повторяющуюся версию поиска решения дилеммы заключенного. Будут ли Алиса и Боб действовать совместно и откажутся свидетельствовать друг против друга, зная о том, что им придется снова встретиться? Ответ зависит от деталей их соглашения. Например, предположим, Алиса и Боб знают, что им придется провести ровно 100 раундов игры в дилемму заключенного. В таком случае оба они знают, что 100-й раунд не будет повторяющейся игрой, т.е. что ее результат не сможет оказывать влияния на будущие раунды, и поэтому оба они выберут в этом раунде доминантную стратегию *testify*. Но как только будет определен результат 100-го

раунда, 99-й раунд перестанет оказывать влияние на последующие раунды, поэтому в нем также будет достигаться равновесие доминантной стратегии в случае выбора действий (*testify, testify*). По индукции оба игрока должны выбирать действие *testify* в каждом раунде, заработав общий срок по 500 лет тюремного заключения на каждого.

Изменяя правила взаимодействия, можно получить другие решения. Например, предположим, что после каждого раунда существует 99% шансов, что игроки снова встретятся. В таком случае ожидаемое число раундов все еще остается равным 100, но ни один из игроков не знает точно, какой раунд будет последним. При таких условиях возможно поведение, характеризующееся большей степенью сотрудничества. Например, одной из стратегий равновесия для каждого игрока является выбор действия *refuse*, если другой игрок никогда не выбирал действие *testify*. Такую стратегию можно назвать **вечным наказанием**. Предположим, что оба игрока приняли данную стратегию и это известно им обоим. В таком случае, при условии, что ни один из игроков не выберет действие *testify*, в любой момент времени ожидаемое суммарное вознаграждение в будущем для каждого игрока составляет следующее:

$$\sum_{t=0}^{\infty} 0.99^t \cdot (-1) = -100$$

Игрок, который выберет *testify*, получит 0 очков вместо -1 в каждом следующем ходе, но его суммарное ожидаемое будущее вознаграждение становится равным:

$$0 + \sum_{t=0}^{\infty} 0.99^t \cdot (-10) = -99$$

Поэтому на каждом этапе игроки не имеют стимула, который заставил бы их отказаться от стратегии (*refuse, refuse*). Вечное наказание — это стратегия “взаимно гарантированного уничтожения” в рамках дилеммы заключенного: как только один из игроков решит выполнить действие *testify*, он обеспечит получение обоими игроками больших неприятностей. Но такая перспектива развития событий может стать предостережением, только если другой игрок знает, что вы приняли эту стратегию или по меньшей мере что вы могли ее принять.

Существуют и другие стратегии в этой игре, которые являются не такими бескомпромиссными. Наиболее известная из них, называемая **отплатой “зуб за зуб”**, предусматривает, что игрок начинает с действия *refuse*, а затем повторяет предыдущий ход другого игрока во всех последующих ходах. Поэтому Алиса должна отказываться свидетельствовать против Боба до тех пор, пока Боб отказывается свидетельствовать против нее, затем должна выбирать в своем ходе дачу показаний против Боба, как только Боб даст показания против нее, но должна возвращаться к отказу от дачи показаний, после того как это сделает Боб. Хотя эта стратегия очень проста, оказалось, что она является в высшей степени надежной и эффективной в противодействии самым разнообразным стратегиям.

Кроме того, другие решения могут быть получены в результате модификации самих агентов, а не изменения правил их взаимодействия. Предположим, что агенты представляют собой конечные автоматы с *n* состояниями и играют в игру с общим

количество ходов  $m > n$ . Поэтому агенты в какой-то момент становятся неспособными представить целый ряд оставшихся ходов и должны рассматривать их как неизвестные. Это означает, что они не могут выполнять логический вывод по индукции и вправе переходить к наиболее благоприятному равновесию (*refuse, refuse*). В таком случае глупость идет на пользу или, скорее, идет на пользу мнение противника о том, что вы глупы. Ваш успех в таких повторяющихся играх зависит от мнения о вас другого игрока как о тупице или простаке, а не от ваших фактических характеристик.

Полное описание повторяющихся игр, которые представляют собой большую и важную научную область, выходит за рамки данной книги, но они возникают во многих ситуациях. Например, последовательную игру можно организовать, поместив двух агентов в мир  $4 \times 3$ , показанный на рис. 17.1. Если будет определено, что не должно происходить никакого движения, если два агента пытаются одновременно перейти в один и тот же квадрат (эта проблема часто возникает на многих пересечениях разных направлений дорожного движения), то некоторые чистые стратегии могут привести к бесконечному тупику. Одно из решений состоит в том, чтобы для каждого агента был рандомизирован выбор между движением вперед и остановкой на месте; патовая ситуация будет быстро разрешена и оба агента смогут продолжить свое движение. Именно такой подход применяется при разрешении коллизий между пакетами в сетях Ethernet.

Известные в настоящее время методы выработки решений для повторяющихся игр напоминают методы, применяемые в играх с поочередными ходами, описанных в главе 6, в том, что дерево игры может быть сформировано от корня вниз и решено от листьев вверх. Основное различие между ними состоит в том, что вместо простого определения максимума или минимума дочерних значений алгоритм должен найти решение игры в терминах смешанных стратегий на каждом уровне, при условии, что для дочерних узлов найдены решения и имеются точно определенные значения, с которыми можно дальше работать.

Повторяющиеся игры в частично наблюдаемых вариантах среды называются играми с **частичной информацией**. К примерам таких игр относятся карточные игры наподобие покера и бриджа, в которых каждый игрок видит только некоторое подмножество карт, а также более серьезные “игры”, такие как моделирование атомной войны, когда ни одна из сторон не знает местонахождение всех пусковых установок противника. Решения игр с частичной информацией можно найти, рассматривая дерево доверительных состояний, как и в случае задач POMDP (см. раздел 17.4). Одно важное различие между ними состоит в том, что собственное доверительное состояние является наблюдаемым, а доверительное состояние противника — нет. Для таких игр практически применимые алгоритмы были разработаны только недавно. Например, найдено решение для некоторой упрощенной версии покера и доказано, что вариант, в котором игроки блефуют, действительно является рациональным, по крайней мере в составе тщательно сбалансированной смешанной стратегии. В результате этих исследований было сделано одно важное открытие, которое заключается в том, что смешанные стратегии полезны не только для того, чтобы сделать действия игрока непредсказуемыми, но и для минимизации объема информации, которую противник может извлечь из наблюдений за действиями этого игрока. Интересно отметить, что разработчики программ для игры в бридж хорошо понимают важность сбора и сокрытия информации, но ни один из них не предложил использовать рандомизированные стратегии.

До сих пор обнаруживались определенные барьеры, которые препятствовали широкому использованию теории игр в проектах агентов. Во-первых, следует отметить, что при поиске решения на основе равновесия Нэша игрок предполагает, что его противник со всей определенностью будет вести игру на основе равновесной стратегии. Это означает, что игрок не способен учесть какие-либо убеждения, которые у него могут быть в отношении того, как скорее всего будут действовать другие игроки, и поэтому не сможет воспользоваться своими важными преимуществами, защищаясь от угроз, которые так никогда и не материализуются. Эта проблема частично решается благодаря использованию понятия **равновесия Байеса–Нэша**, т.е. равновесия применительно к распределению априорных вероятностей игрока по отношению к стратегиям других игроков; иными словами, равновесие Байеса–Нэша выражает уверенность игрока в том, какие вероятные стратегии будут применять другие игроки. Во-вторых, в настоящее время отсутствует удобный способ совместного применения стратегий управления, основанных на теории игры и модели РОМДР. Из-за наличия этих и других проблем теория игр в основном использовалась для анализа вариантов среды, находящихся в равновесном состоянии, а не для управления агентами, действующими в некоторой среде. Но ниже будет показано, что теория игр может помочь и при проектировании вариантов среды.

## 17.7. ПРОЕКТИРОВАНИЕ МЕХАНИЗМА

В предыдущем разделе рассматривался вопрос: “Дана некоторая игра; какой является рациональная стратегия?” В этом разделе приведен ответ на другой вопрос: “Предположим, что агенты являются рациональными; какую игру следует для них спроектировать?” А именно, требуется спроектировать некоторую игру, решения которой, состоящие из действий каждого агента, осуществляющего свою собственную рациональную стратегию, приводят к максимизации некоторой глобальной функции полезности. Эта проблемная область называется **проектированием механизма**, а иногда **обратной теорией игр**. Проектирование механизма представляет собой фундамент экономики и политологии. Применительно к коллекциям агентов этот подход обеспечивает возможность использования механизмов теории игр для создания успешно действующих систем из совокупности более ограниченных системных компонентов (даже противоборствующих системных компонентов), во многом аналогично тому, как создаются команды людей, способные достигать целей, далеко превосходящих возможности отдельно взятого человека.

К примерам применения принципов проектирования механизма относится организация аукционов по распродаже дешевых авиабилетов, маршрутизация пакетов TCP между компьютерами, принятие решения о том, как следует распределять выпускников медицинских учебных заведений по лечебным учреждениям, а также определение способа взаимодействия роботизированных игроков-футболистов со своими капитанами команд. Проектирование механизма вышло за рамки академической тематики в конце 1990-х годов, когда несколько государств, столкнувшись с проблемой аукционной распродажи лицензий на вещание в различных частотных диапазонах, потеряли сотни миллионов долларов потенциальных доходов в результате плохого проектирования механизма. Формально **механизм** состоит, во-первых, из языка для описания (потенциально бесконечного) множества допустимых стратегий, которыми могут ру-

ководствоваться агенты, и, во-вторых, из правила определения результата  $G$ , которое регламентирует вознаграждения, получаемые агентами при наличии некоторого профиля стратегий, состоящего из допустимых стратегий.

На первый взгляд задача проектирования механизма может показаться тривиальной. Предположим, что глобальная функция полезности  $U$  декомпонуется на любое множество функций полезности отдельных агентов  $U_i$ , такое, что  $U = \sum_i U_i$ . В таком случае можно утверждать, что если каждый агент максимизирует свою собственную полезность, то нет сомнения в том, что это автоматически приведет к максимизации глобальной полезности. (Например, в лекциях по начальному курсу капитализма утверждается, что если каждый старается стать богаче, то общее благосостояние общества повышается.) К сожалению, такой принцип не оправдывается. Действия каждого агента могут повлиять на благосостояние других агентов так, что глобальная полезность уменьшится. Одним из примеров этого является **трагическая деградация общих пастбищ** (tragedy of the commons) — ситуация, в которой отдельные фермеры сгоняют все свои стада для того, чтобы они бесплатно паслись на общих сельских пастбищах, в результате чего наступает деградация этих общих пастбищ, что приводит к достижению отрицательной полезности для всех фермеров. Каждый фермер, отдельно взятый, действовал рационально, рассуждая, что он может использовать общие пастбища бесплатно, и хотя интенсивное использование общих пастбищ может привести к их деградации, его отказ выгонять на них свой скот делу не поможет (поскольку другие от этого все равно не откажутся). Аналогичные доводы выдвигаются теми, кто не желает ограничивать выбросы промышленных загрязнений в атмосферу и в океаны.

Стандартный подход при проектировании механизма для решения подобных проблем состоит в том, что с каждого агента должна взиматься плата за использование общих пастбищ. Вообще говоря, следует обеспечить, чтобы все **побочные последствия** (отрицательные влияния на глобальную полезность, которые не сказываются на результатах, полученных отдельными агентами) были выражены явно. При решении задачи труднее всего правильно определить цены. В пределе этот подход сводится к созданию механизма, который действительно требует от каждого агента, чтобы он максимизировал глобальную полезность. Применительно к отдельному агенту, который не обладает возможностью оценивать текущее состояние мира и не может наблюдать за результатами действий всех других агентов, такая задача становится невероятно трудной. Поэтому проектирование механизма должно сосредоточиваться на поиске таких механизмов, при использовании которых задача принятия решений для отдельных агентов становится несложной.

Вначале рассмотрим аукционы. В своей наиболее общей форме аукцион представляет собой механизм продажи некоторых товаров членам определенного сообщества покупателей. Стратегиями являются стратегии ведения торгов, а результаты определяют, кто получит товары и сколько заплатит. Одним из примеров, в которых аукционы могут войти в состав тематики искусственного интеллекта, является принятие решения совокупностью агентов о том, будут ли они участвовать в совместном плане. Хансбергер и Грош [706] показали, что эта задача может быть эффективно решена с помощью аукциона, в котором агенты распределяют между собой роли в совместном плане.

В данном разделе рассмотрим аукционы, в которых, во-первых, имеется только один вид товара, во-вторых, каждый покупатель руководствуется собственным значением полезности  $v_i$  по отношению к данному товару, и, в-третьих, эти значения известны только покупателю. Покупатели вносят свои предложения  $b_i$ , а товары передаются тому, кто сделал предложение с самой высокой ценой, но механизм определяет, как нужно делать эти предложения и какую цену платит победитель (она не обязательно должна быть равна  $b_i$ ). Наиболее широко известным типом аукциона является **английский аукцион**, в котором лицо, проводящее аукцион, повышает цену товара, проверяя, остались ли еще заинтересованные покупатели, до тех пор, пока не останется только один потенциальный покупатель. Этот механизм обладает тем свойством, что покупатель, руководствуясь наивысшим значением  $v_i$ , получает товары по цене  $b_m + d$ , где  $b_m$  — наивысшая предложенная цена среди цен, предложенных всеми другими игроками, а  $d$  — величина, на которую лицо, проводящее аукцион, наращивает цену от одного предложения к другому<sup>9</sup>. Поэтому английский аукцион обладает тем свойством, что участники аукциона руководствуются простой доминантной стратегией — продолжать выдвигать предложения до тех пор, пока текущая цена остается ниже назначенного лично вами значения. Напомним, что “доминантной” стратегией называется стратегия, позволяющая противодействовать всем другим стратегиям, а это, в свою очередь, имеет тот смысл, что игрок может ею руководствоваться, невзирая на то, что существуют любые другие стратегии. Поэтому игроки не обязаны терять время и энергию, пытаясь предугадать возможные стратегии других игроков. Механизм, в котором игроки имеют доминантную стратегию, позволяющую скрывать свои истинные побуждения, называется механизмом **защиты стратегии**.

Одним из отрицательных свойств английского аукциона являются большие затраты на связь, поэтому либо следует проводить аукцион в одном помещении, либо предоставить всем участникам аукциона быстродействующие, надежные линии связи. Альтернативным механизмом, для которого требуется гораздо меньший объем связи, является **аукцион с запечатанными предложениями**. В этом случае каждый участник аукциона подготавливает единственное предложение и сообщает его лицу, проводящему аукцион; после сбора всех предложений побеждает предложение с наибольшей ценой. При использовании этого механизма стратегия, в которой участник аукциона выдвигает в качестве предложения то истинное значение, которым он руководствуется, больше не является доминантной. Если значение полезности для игрока равно  $v_i$  и он считает, что максимальным из предложений всех других игроков будет  $b_m$ , то он должен выдвинуть предложение с ценой, меньшей  $v_i$  и равной  $b_m + \epsilon$ . Двумя недостатками аукциона с запечатанными предложениями является то, что игрок с наивысшим значением полезности  $v_i$  может не получить товары, а также то, что игроки должны затрачивать свои усилия на прогнозирование стратегий других игроков.

В результате небольшого изменения правил аукционов с запечатанными предложениями был разработан **аукцион с запечатанными предложениями на вторую по**

<sup>9</sup> Фактически шансы на то, что игрок с наибольшим значением  $v_i$  не сможет получить товары, невелики; это произойдет, только если  $b_m < v_i < b_m + d$ . Вероятность такого события может стать сколь угодно малой благодаря уменьшению приращения  $d$ .

**величине цену**, называемый также **аукционом Викри**<sup>10</sup>. При проведении таких аукционов победитель платит цену, указанную во втором по величине предложении, а не цену, указанную в своем собственном предложении. Эта простая модификация позволяет полностью устраниТЬ необходимость в проведении сложных рассуждений, которые требовались в стандартных аукционах с запечатанными предложениями (т.е. в аукционах с **первой по величине ценой**), поскольку теперь доминантная стратегия состоит в том, чтобы выдвинуть в качестве предложения цену, соответствующую фактической собственной величине полезности. Чтобы убедиться в этом, достаточно отметить, что каждый игрок может рассматривать данный аукцион как игру с двумя игроками, игнорируя всех игроков, кроме себя и лица, выдвинувшего предложение с наибольшей ценой среди всех других игроков. Значение полезности для игрока  $i$ , выраженное в виде цены, указанной в его предложении,  $b_i$ , само значение полезности  $v_i$ , которым он руководствуется, и наилучшее предложение среди других игроков,  $b_m$ , связаны между собой таким соотношением:

$$u_i(b_i, b_m) = \begin{cases} (v_i - b_m) & \text{если } b_i > b_m \\ 0 & \text{в противном случае} \end{cases}$$

Чтобы убедиться в том, что выбор значения  $b_i = v_i$  представляет собой доминантную стратегию, отметим, что при положительном значении  $(v_i - b_m)$  любое предложение, которое побеждает в аукционе, является оптимальным, и, в частности, аукцион можно выиграть, выдвинув предложение  $v_i$ . С другой стороны, когда значение  $(v_i - b_m)$  является отрицательным, оптимальным становится любое предложение, которое проигрывает в этом аукционе, и, в частности, в аукционе проигрывает предложение, в котором применяется значение  $v_i$ . Поэтому выдвижение предложения на основе значения  $v_i$  представляет собой оптимальную стратегию при всех возможных значениях  $b_m$ , и действительно предложение на основе  $v_i$  — это единственное предложение, которое обладает таким свойством. Благодаря его простоте и минимальным вычислительным требованиям как к тем, кто предоставляет услуги, так и к тем, кто на них претендует, аукцион Викри широко используется при проектировании распределенных систем на основе искусственного интеллекта.

Теперь рассмотрим проблему маршрутизации в Internet. Игроки соответствуют ребрам в графе сетевых соединений. Каждый игрок знает стоимость  $c_i$  передачи сообщения по его собственному ребру, а стоимость того, что сообщение не будет передано для отправки, равна 0. Задача состоит в том, чтобы найти самый дешевый путь передачи сообщения от отправителя к получателю, где стоимость всего маршрута представляет собой сумму стоимостей отдельных ребер. В главе 4 приведено несколько алгоритмов вычисления кратчайшего пути с учетом стоимости ребер, поэтому остается только предусмотреть, чтобы каждый агент сообщал свою истинную стоимость  $c_i$ . К сожалению, если будет просто передан запрос каждому агенту, он сообщит стоимости, которые слишком велики, чтобы вынудить нас отправлять свои сообщения по каким-то другим каналам. Поэтому необходимо разработать механизм защиты стратегии. Один из таких механизмов состоит в том, чтобы выплачивать каждому игроку вознаграждение  $p_i$ , равное длине кратчайшего пути, который

---

<sup>10</sup> Названный в честь Уильяма Викри (1914–1996), лауреата Нобелевской премии по экономике за 1996 год.

не включает  $i$ -е ребро, за вычетом длины кратчайшего пути (вычисленной с помощью алгоритма поиска), где предполагается, что стоимость  $i$ -го ребра равна 0:

$$p_i = \text{Length}(\text{путь}, \text{ где } c_i = \infty) - \text{Length}(\text{путь}, \text{ где } c_i = 0)$$

Можно показать, что при использовании такого механизма доминантной стратегией для каждого игрока становится выдача правильных сведений о значении  $c_i$  и что применение такого подхода приводит к получению самого дешевого пути. Но, несмотря на это желательное свойство, описанный механизм не используется на практике из-за высокой стоимости связи и централизованных вычислений. Проектировщик механизма должен связаться со всеми  $n$  игроками, а затем решить задачу оптимизации. Применение такого подхода имело бы смысл, если бы затраты можно было амортизировать по многим сообщениям, но в реальной сети стоимости  $c_i$  непрерывно колеблются из-за заторов в каналах, а также из-за того, что некоторые компьютеры завершают свою работу аварийно, а другие, наоборот, переходят в оперативный режим. Полностью удовлетворительное решение этой проблемы еще не было предложено.

## 17.8. РЕЗЮМЕ

В этой главе показано, как использовать знания о мире для принятия решений, даже если результаты действий являются неопределенными, а вознаграждения за действия могут оставаться недоступными до тех пор, пока не будет осуществлен целый ряд действий. Основные идеи этой главы кратко изложены ниже.

- Задачи последовательного принятия решений в неопределенных вариантах среды, называемые также **марковскими процессами принятия решений** (Markov Decision Process — MDP), определяются с помощью **моделей перехода**, дающих вероятностные результаты действий, и **функции вознаграждения**, которая показывает, какое вознаграждение соответствует каждому состоянию.
- Полезность последовательности состояний представляет собой сумму всех вознаграждений вдоль этой последовательности, которая, возможно, со временем подвергается обесцениванию. Решением задачи MDP является **стратегия**, в которой с каждым состоянием, достижимым для агента, связано некоторое решение. Оптимальная стратегия максимизирует полезность встречающейся последовательности состояний при ее осуществлении.
- Полезностью состояния является ожидаемая полезность последовательностей состояний, встречающихся при осуществлении оптимальной стратегии, начиная с этого состояния. Алгоритм **итерации по значениям** для решения задач MDP действует по принципу итеративного решения уравнений, связывающих полезности каждого состояния с полезностями его соседних состояний.
- В алгоритме **итерации по стратегиям** чередуются этап вычисления полезностей состояний согласно текущей стратегии и этап усовершенствования текущей стратегии по отношению к текущим полезностям.
- Задачи MDP в частично наблюдаемой среде, или задачи POMDP, являются гораздо более трудными для решения, чем задачи MDP. Они могут быть ре-

шены путем преобразования в задачу MDP в непрерывном пространстве доверительных состояний. Оптимальное поведение при решении задач POMDP должно предусматривать сбор информации для уменьшения неопределенности и поэтому принятия лучших решений в будущем.

- Для вариантов среды POMDP может быть создан агент, действующий на основе теории решений. В таком агенте для представления модели перехода и модели наблюдения для обновления его доверительного состояния и проектирования возможных последовательностей действий в прямом направлении используется **динамическая сеть принятия решений**.
- **Теория игр** описывает рациональное поведение для агентов в тех ситуациях, в которых одновременно взаимодействуют множество агентов. Решениями для игр являются **равновесия Нэша** — профили стратегий, в которых ни один из агентов не имеет стимулов, под влиянием которых он мог бы уклониться от определенной для него стратегии.
- **Проектирование механизма** может использоваться для определения правил, по которым должно быть организовано взаимодействие агентов в целях максимизации некоторой глобальной полезности благодаря функционированию отдельных рациональных агентов. Иногда удается найти механизмы, позволяющие достичь этой цели, не требуя от каждого агента, чтобы он учитывал то, какие варианты выбраны другими агентами.

Мы вернемся к тематике задач MDP и POMDP в главе 21, где описаны методы **обучения с подкреплением**, позволяющие агенту совершенствовать свое поведение на основании опыта, полученного в последовательных, неопределенных вариантах среды.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Основателем современного подхода к анализу задач последовательного принятия решений был Ричард Беллман [97], который в целом предложил использовать подход на основе динамического программирования и в частности алгоритм итерации по значениям. В тезисах докторской диссертации Рона Говарда [691] введено понятие итерации по стратегиям и изложена идея среднего вознаграждения, предназначенная для применения при решении задач с бесконечным горизонтом. Несколько дополнительных результатов было предложено в [96]. Модифицированный алгоритм итерации по стратегиям описан в [1245] и [1534]. Алгоритм асинхронной итерации по стратегиям был проанализирован Уильямсом и Бердом [1598], которые также доказали свойство граничной убыточности стратегии, рассматриваемое в уравнении 17.9. Результаты анализа обесценивания с точки зрения стационарных предпочтений приведены в [834]. В [120], [121] и [1244] дано строгое введение в проблематику задач последовательного принятия решений. В [1170] описаны результаты исследования вычислительной сложности задач MDP.

Важную роль в ознакомлении сообщества разработчиков в области искусственного интеллекта с задачами MDP сыграли оригинальные работы Саттона [1477] и Уоткинса [1558] по применению методов обучения с подкреплением для решения задач MDP, а также вышедший немного позже обзор [74]. (В более ранней работе

[1580] содержались во многом аналогичные идеи, но они не были развиты до такой же степени.) Связь между задачами MDP и задачами планирования в искусственном интеллекте была впервые показана Свеном Кёнигом [817], который также продемонстрировал, что вероятностные операторы Strips позволяют создавать компактные представления для моделей перехода (см. также [1575]). В [359] и [1492] сделана попытка преодолеть комбинаторную сложность больших пространств состояний с использованием ограниченного горизонта поиска и абстрактных состояний. Эвристики, основанные на стоимости информации, могут использоваться для определения в пространстве состояний таких областей, где локальное расширение горизонта приводит к значительному повышению качества решения. Агенты, применяющие такой подход, могут соразмерять свои усилия под давлением дефицита времени и вырабатывать некоторые варианты поведения (такие как использование знакомых “проторенных тропинок”) для быстрого поиска путей через пространство состояний без необходимости повторно вычислять в каждой точке оптимальные решения. В опубликованных недавно работах Бутильера и др. [158], [159] основные усилия сосредоточены на динамическом программировании с использованием символьических представлений моделей перехода и функций стоимости на основе формул propositionalной логики и логики первого порядка.

В [47] впервые было показано, что задачи MDP в частично наблюдаемой среде могут быть преобразованы в обычные задачи MDP с использованием доверительных состояний. Первый полный алгоритм точного решения для марковских процессов принятия решений в частично наблюдаемой среде (Partially-Observable Markov Decision Process — POMDP) был предложен Эдвардом Сондиком [1446] в его докторской диссертации (опубликованная позднее журнальная статья Смоллвуда и Сондика [1433] содержит некоторые ошибки, но является более доступной). В [946] приведен обзор состояния разработок в области решения задач POMDP. Первым значительным вкладом в рамках искусственного интеллекта был алгоритм Witness [227], [758] — усовершенствованная версия алгоритма итерации по значениям, применяемого для решения задач POMDP. Вслед за ним вскоре были разработаны другие алгоритмы, в том числе основанные на подходе, предложенном Хансеном [612], который предусматривает инкрементное определение стратегии с помощью конечного автомата. В представлении стратегии с помощью конечного автомата доверительные состояния непосредственно соответствуют конкретным состояниям автомата. Приближенно оптимальные стратегии для задач POMDP могут формироваться с помощью прямого поиска, применяемого в сочетании с формированием выборок из возможных результатов наблюдений и действий [784], [1134]. Другие результаты исследований в области алгоритмов POMDP описаны в главе 21.

Основные идеи по созданию архитектуры агента с использованием динамических сетей принятия решений были предложены в [360]. В книге *Planning and Control* Дина и Уэллмана [363] этот подход развит намного глубже и установлена связь между моделями DBN/DDN и классической литературой теории управления, посвященной вопросам фильтрации. Татмен и Шахтер [1497] показали, как применять алгоритмы динамического программирования к моделям DDN. В [1325] описаны различные способы обеспечения применения таких агентов в более широких масштабах и указан ряд нерешенных исследовательских проблем.

Корни теории игр можно проследить до предложений по изучению конкурентных и кооперативных взаимодействий людей с помощью научного и математиче-

ского подхода, которые были выдвинуты еще в XVII столетии Христианом Гюйгенсом и Готфридом Лейбницем. На протяжении XIX столетия некоторые ведущие экономисты создавали простые математические модели для анализа конкретных примеров конкурентных ситуаций. Первые формальные результаты в области теории игр принадлежат Цермело [1641], который за год до этого предложил некоторую форму минимаксного поиска для игр, хотя и неправильную. Эмиль Борель [152] ввел понятие смешанной стратегии. Джон фон Нейман [1545] доказал, что любая игра с двумя участниками и нулевой суммой имеет максиминное равновесие в смешанных стратегиях и полностью определенное значение. Сотрудничество Джона фон Неймана с экономистом Оскаром Моргенштерном привело к публикации в 1944 году книги *Theory of Games and Economic Behavior* (Теория игр и экономического поведения) [1546], которая сыграла решающую роль в создании теории игр. Публикация этой книги задерживалась из-за нехватки бумаги во время войны до тех пор, пока один из членов семейства Рокфеллеров не субсидировал публикацию.

В 1950 году в возрасте 21 года Джон Нэш опубликовал свои идеи, касающиеся достижения равновесия в играх общего типа [1112]. Его определение равновесного решения получило известность под названием *равновесия Нэша* (хотя впервые появилось в работе Курно [297]). После длительной задержки из-за шизофрении, которой он страдал с 1959 года, Нэш получил Нобелевскую премию по экономике (наряду с Рейнхартом Селтеном и Джоном Харсаны) в 1994 году.

Равновесие Байеса–Нэша описано в [622] и обсуждается в [757]. Некоторые проблемы использования теории игр для управления агентами рассматриваются в [129].

Дilemma заключенного была разработана для использования в качестве учебного упражнения Альбертом В. Такером в 1950 году и тщательно исследована в [50]. Понятие повторяющихся игр было представлено в [963], а игры с частичной информацией — в [862]. Первый практически применимый алгоритм для игр с частичной информацией был разработан в рамках искусственного интеллекта Коллером и др.[821]; в статье [822] дано удобное для чтения введение в эту общую область и описана действующая система представления и решения последовательных игр. Теория игр и задач MDP объединена в теорию марковских игр [937]. Шепли [1398] фактически описал алгоритм итерации по значениям до Беллмана, но его результаты не получили широкого признания, вероятно, потому, что были представлены в контексте марковских игр. К числу основных учебников по теории игр относятся [510], [1109] и [1161].

Задача, послужившая стимулом к созданию области проектирования механизма, трагическая деградация общих паства, была представлена в [619]. Гурвич [709] создал математические основы для проектирования механизма. Милгром [1048] описал разработанный им механизм аукциона, который охватывает диапазон сделок в несколько миллиардов долларов. Понятие аукционов может также использоваться в планировании [706] и составлении расписаний [1267]. В [1539] приведен краткий обзор тематики аукционов в связи с публикациями по компьютерным наукам, а в [1307] представлено описание способов применения этого подхода для решения распределенных задач искусственного интеллекта, занявшее целую книгу. Родственные работы по проблематике распределенных задач искусственного интеллекта публикуются также под другими названиями, включая коллективный интеллект [1516] и управление на основе рынка [269]. Статьи по вычислительным проблемам, связанным с аукционами, часто публикуются в трудах конференции *ACM Conferences on Electronic Commerce*.

## УПРАЖНЕНИЯ

- 17.1. Для мира  $4 \times 3$  (см. рис. 17.1) рассчитайте, какие квадраты могут быть достигнуты из квадрата  $(1, 1)$  с помощью последовательности действий  $[Up, Up, Right, Right, Right]$  и с какими вероятностями. Объясните, как эти вычисления связаны с задачей проектирования скрытой марковской модели.
- 17.2. Предположим, что в качестве полезности последовательности состояний определено максимальное вознаграждение, полученное в любом из состояний этой последовательности. Покажите, что данная функция полезности не приводит к формированию стационарных предпочтений между последовательностями состояний. Остается ли возможным определение на состояниях такой функции полезности, что принятие решений на основе полезности MEU позволит сформировать оптимальное поведение?
- 17.3. Может ли любая задача конечного поиска быть точно преобразована в марковскую задачу принятия решений, такую, что оптимальное решение последней является также оптимальным решением первой? Если да, то объясните, как преобразовать эту задачу и как выполнить обратное преобразование решения, а если нет, объясните, почему ваш ответ отрицателен (в частности, приведите контрпример).
- 17.4. Рассмотрите задачу MDP без обесценивания, имеющую три состояния,  $(1, 2, 3)$ , с вознаграждениями  $-1, -2, 0$  соответственно. Состояние 3 является терминальным. В состояниях 1 и 2 имеются два возможных действия,  $a$  и  $b$ . Модель перехода описана ниже.
  - В состоянии 1 действие  $a$  переводит агента в состояние 2 с вероятностью 0,8 и оставляет агента в том же состоянии с вероятностью 0,2.
  - В состоянии 2 действие  $a$  переводит агента в состояние 1 с вероятностью 0,8 и оставляет агента в том же состоянии с вероятностью 0,2.
  - Либо в состоянии 1, либо в состоянии 2 действие  $b$  переводит агента в состояние 3 с вероятностью 0,1 и оставляет агента в том же состоянии с вероятностью 0,9.

Дайте ответы на приведенные ниже вопросы.

- a) Какие характеристики оптимальной стратегии в состояниях 1 и 2 могут быть определены качественно?
- б) Примените алгоритм итерации по стратегиям для определения оптимальной стратегии и значений состояний 1 и 2, полностью демонстрируя каждый этап. Примите предположение, что исходная стратегия включает действие  $b$  в обоих состояниях.
- в) Как повлияет на работу алгоритма итерации по стратегиям включение в исходную стратегию действия  $a$  в обоих состояниях? Поможет ли применение обесценивания? Зависит ли оптимальная стратегия от коэффициента обесценивания?

- 17.5.** Иногда задачи MDP формулируются на основе функции вознаграждения  $R(s, a)$ , которая зависит от выполненного действия, или функции вознаграждения  $R(s, a, s')$ , которая зависит также от результирующего состояния.
- Запишите уравнения Беллмана для этих формулировок.
  - Покажите, как можно преобразовать задачу MDP с функцией вознаграждения  $R(s, a, s')$  в другую задачу MDP с функцией вознаграждения  $R(s, a)$ , такую, что оптимальные стратегии в новой задаче MDP будут точно соответствовать оптимальным стратегиям в первоначальной задаче MDP.
  - Выполните аналогичные действия для преобразования задачи MDP с функцией  $R(s, a)$  в задачу MDP с функцией  $R(s)$ .
- 17.6.** Рассмотрим мир  $4 \times 3$ , показанный на рис. 17.1.
- Реализуйте имитатор для этой среды, такой, чтобы можно было легко изменять географию данной среды. Некоторый код для решения указанной задачи уже находится в оперативном репозитории кода.
  - Создайте агента, в котором используется итерация по стратегиям, и измерьте его производительность в имитаторе среды, начиная с различных начальных состояний. Выполните несколько экспериментов из каждого начального состояния и сравните среднее суммарное вознаграждение, полученное за каждый прогон, с полезностью состояния, которая определяется применяемым вами алгоритмом.
  - Проведите эксперименты в среде с увеличенными размерами. Как изменяется время прогона алгоритма итерации по стратегиям в зависимости от размеров этой среды?
- 17.7.** Для среды, показанной на рис. 17.1, найдите все пороговые значения функции полезности  $R(s)$ , такие, что после пересечения этого порога изменяется оптимальная стратегия. Вам потребуется способ вычисления оптимальной стратегии и ее значения для фиксированной функции  $R(s)$ . (*Подсказка.* Докажите, что значение любой фиксированной стратегии линейно зависит от функции  $R(s)$ .)
- 17.8.** В этом упражнении рассматриваются задачи MDP с двумя игроками, которые соответствуют играм с нулевой суммой и поочередными ходами, подобным описанным в главе 6. Допустим, что игроки обозначены как  $A$  и  $B$ , и предположим, что  $R(s)$  — вознаграждение для игрока  $A$  в состоянии  $s$ . (Вознаграждение для игрока  $B$  всегда равно ему противоположно.)
- Допустим, что  $U_A(s)$  — полезность состояния  $s$ , когда очередь хода в состоянии  $s$  принадлежит игроку  $A$ ;  $U_B(s)$  — полезность состояния  $s$ , когда очередь хода в состоянии  $s$  принадлежит игроку  $B$ . Все вознаграждения и полезности вычисляются с точки зрения игрока  $A$  (как и в минимаксном дереве игры). Запишите уравнения Беллмана, определяющие  $U_A(s)$  и  $U_B(s)$ .
  - Объясните, как осуществить итерацию по значениям для двух игроков с помощью этих уравнений, и определите подходящий критерий останова.
  - Рассмотрите игру, приведенную на рис. 6.12. Изобразите пространство состояний (а не дерево игры), показывая ходы игрока  $A$  сплошными ли-

ниями, а ходы игрока  $B$  — штриховыми линиями. Обозначьте каждое состояние значением  $R(s)$ . Вы обнаружите, что удобно расположить состояния  $(s_A, s_B)$  в виде двухмерной решетки, используя в качестве “координат” значения  $s_A$  и  $s_B$ .

- г) Теперь примените итерацию по значениям для двух игроков, чтобы найти решение этой игры, и определите оптимальную стратегию.

**17.9.** Покажите, что любое равновесие доминантной стратегии представляет собой равновесие Нэша, но обратное утверждение не верно.

**17.10.** В детской игре в камень, бумагу и ножницы каждый игрок в какое-то время пытается выяснить, кто выбрал камень, бумагу или ножницы. Бумага обертывает камень, камень тупит ножницы, а ножницы режут бумагу. В расширенной версии игры в камень, бумагу, ножницы, огонь и воду приняты следующие условия: огонь побеждает камень, бумагу и ножницы; камень, бумага и ножницы побеждают воду; вода побеждает огонь. Запишите матрицу вознаграждений и найдите решение со смешанной стратегией для этой игры.

**17.11.** Найдите решение для игры в чет и нечет на трех пальцах.

**17.12.** До 1999 года команды Национальной хоккейной лиги получали 2 очка за победу, 1 за ничью и 0 за поражение. Можно ли рассматривать хоккей с такими правилами как игру с постоянной суммой? В 1999 году правила были изменены таким образом, что команда получает 1 очко за проигрыш в дополнительное время. Победившая команда все еще получает 2 очка. Изменится ли с учетом этого дополнения ответ на вопрос, приведенный выше? Если бы за это не было наказания, то было бы рациональным для двух команд втайне договариваться, чтобы оканчивать отборочную игру ничьей, а затем доводить ее до результативного завершения в дополнительное время? Предполагается, что полезность для каждой команды измеряется количеством полученных ею очков и что обоюдно известна априорная вероятность  $p$  того, что первая команда победит в дополнительное время. При каких значениях  $p$  обеим командам следовало бы заключать такое соглашение?

**17.13.** Приведенная в табл. 17.4 матрица вознаграждений, впервые опубликованная Блиндером [137] и проанализированная Бернштейном [114], показывает, какую игру ведут между собой конгрессмены-политики (сокращенно Pol) и руководство Федеральной резервной системы (сокращенно Fed).

Таблица 17.4. Матрица вознаграждений в административной игре

	Fed: сузить	Fed: ничего не делать	Fed: расширить
Pol: сузить	F=7, P=1	F=9, P=4	F=6, P=6
Pol: ничего не делать	F=8, P=2	F=5, P=5	F=4, P=9
Pol: расширить	F=3, P=3	F=2, P=7	F=1, P=8

Конгрессмены могут расширять или сужать налоговую политику, а руководители Федеральной резервной системы могут расширять или сужать бюджетную политику (и, безусловно, та и иная сторона может выбрать вариант, в котором не предусматривается внесение каких-либо изменений.) Кроме того, каждая из этих сторон имеет определенные предпочтения в отношении того,

кто и что должен делать, поскольку ни те ни другие не хотят вызвать недовольство населения. Показанные выше вознаграждения представляют собой ранги упорядочения от 9 до 1, т.е. от первого варианта до последнего. Найдите равновесие Нэша для этой игры в рамках чистых стратегий. Является ли это решение оптимальным согласно критерию Парето? Рекомендуем читателю проанализировать в свете этих исследований политику администраций США последних созывов.

## Часть VI

### ОБУЧЕНИЕ

Обучение на основе наблюдений	864
Применение знаний в обучении	902
Статистические методы обучения	945
Обучение с подкреплением	1010

# 18 ОБУЧЕНИЕ НА ОСНОВЕ НАБЛЮДЕНИЙ

*В данной главе рассматриваются агенты, способные усовершенствовать свое поведение благодаря тщательному изучению собственного опыта.*

В основе обучения лежит представление о том, что результаты восприятия должны использоваться не только для осуществления действий, но и для повышения способности агента действовать в будущем. Обучение происходит по мере того, как агент наблюдает за своим взаимодействием с миром и собственными процессами принятия решений. Обучение может охватывать широкий спектр действий, начиная от тривиального накопления в памяти результатов полученного опыта, как было показано на примере агента для мира вампира в главе 10, и заканчивая созданием целых научных теорий, что было успешно продемонстрировано Альбертом Эйнштейном. В этой главе рассматривается **индуктивное обучение** на основе наблюдений. В частности, в ней описано, как в процессе обучения формируются простые теории в терминах пропозициональной логики. В ней также приведены результаты теоретического анализа, позволяющие понять принципы индуктивного обучения.

## 18.1. ФОРМЫ ОБУЧЕНИЯ

В главе 2 было показано, что проект обучающегося агента может рассматриваться как состоящий из **производительного элемента**, определяющего, какие действия должны быть выполнены, и **обучающего элемента**, который модифицирует производительный элемент для того, чтобы он вырабатывал лучшие решения (см. рис. 2.7). Исследователи, работающие в области машинного обучения, предложили целый ряд типов обучающих элементов. Для того чтобы разобраться в их работе, целесообразно рассмотреть, как влияет на их проект тот контекст, в котором они должны функционировать. На проект обучающего элемента влияют три описанных ниже аспекта.

- Компоненты производительного элемента, подлежащие обучению.
- Обратные связи, которые могут применяться для обучения этих компонентов.
- Способы представления, используемые для компонентов.

Проведем анализ каждого из этих аспектов по очереди. В данной книге уже было показано, что существует много способов построения производительного элемента для агента. В главе 2 было описано несколько проектов агентов (см. рис. 2.3–2.6). Ниже перечислены компоненты этих агентов.

1. Средства прямого отображения условий (распространяющихся на текущее состояние) в действия.
2. Средства логического вывода релевантных свойств мира из последовательности результатов восприятия.
3. Информация о том, как развивается мир и какие результаты возможных действий могут быть получены агентом.
4. Информация о полезности, которая показывает, насколько желательными являются те или иные состояния мира.
5. Информация о ценности действий, показывающая желательность действий.
6. Цели, описывающие классы состояний, достижение которых максимизирует полезность для агента.

Обучение каждого из этих компонентов может осуществляться с помощью соответствующей обратной связи. Рассмотрим, например, агента, который учится вождению, чтобы стать таксистом. Каждый раз, когда инструктор кричит “Тормози!”, агент должен усвоить очередное правило “условие–действие”, позволяющее определить, когда следует тормозить (компонент 1). Рассматривая множество видеоизображений, на которых, как ему сказано, имеются автобусы, он может научиться распознавать автобусы (компонент 2). Осуществляя попытки выполнения действий и наблюдая за их результатами (например, проводя жесткое торможение на мокрой дороге), он может определить путем обучения, каковы результаты его действий (компонент 3). В дальнейшем, перестав получать чаевые от пассажиров, которые почувствовали себя полностью разбитыми во время утомительной поездки, агент может обучить полезный компонент своей общей функции полезности (компонент 4), который будет подсказывать, что пассажиров надо беречь.

Тип обратной связи, доступной для обучения, обычно является наиболее важным фактором, определяющим характер задачи обучения, с которой сталкивается агент. В области машинного обучения, как правило, различаются три случая: **контролируемое обучение, неконтролируемое обучение и обучение с подкреплением**.

В задаче  **контролируемого обучения** предусматривается изучение некоторой функции на примерах ее входных и выходных данных. Все приведенные выше случаи, касающиеся компонентов 1, 2 и 3, представляют собой примеры задач контролируемого обучения. В случае, который относится к компоненту 1, агент изучает правило “условие–действие”, касающееся торможения, т.е. функцию, которая связывает входные состояния с булевым выходом (тормозить или не тормозить). В случае компонента 2 агент изучает функцию, преобразующую входные изображения в булев выход (который показывает, содержит ли изображение автобус). В случае компонента 3 сведения о торможении выражаются в виде функции, которая связывает

сстояния и действия по торможению, скажем, с длиной тормозного пути в футах. Обратите внимание на то, что в случаях 1 и 2 учитель указывает в примерах правильное выходное значение, а в случае 3 выходное значение должно быть получено непосредственно из результатов восприятия агента. В полностью наблюдаемых вариантах среды всегда соблюдается такое условие, что агент может наблюдать за результатами своих действий и поэтому использовать методы контролируемого обучения, чтобы научиться предсказывать эти результаты. В частично наблюдаемых вариантах среды задача обучения становится более сложной, поскольку непосредственные результаты могут оказаться недоступными для восприятия.

Задача **неконтролируемого обучения** касается выявления определенных закономерностей во входных данных в тех условиях, когда не задаются конкретные выходные значения. Например, агент-водитель такси может постепенно развить представление о “днях с хорошими условиями дорожного движения” и “днях с плохими условиями дорожного движения”, даже не получая обозначенных соответствующими заголовками примеров тех или других дней. Обучающийся агент, полностью остающийся без контроля, не может узнать в процессе обучения, что делать, поскольку он не имеет информации о том, каковым является определение понятия правильного действия или желательного состояния. Мы будем изучать проблему неконтролируемого обучения в основном в контексте систем формирования вероятностных рассуждений (глава 20).

Задача **обучения с подкреплением**, которая будет рассматриваться в главе 21, является наиболее общей из этих трех категорий. Агент, проходящий обучение с подкреплением, не получает от учителя указаний о том, что делать, а должен обучаться на основе **подкрепления**<sup>1</sup>. Например, отсутствие чаевых в конце поездки (или большой штраф за столкновение с идущим впереди автомобилем) будет служить для агента определенным показателем того, что его поведение нежелательно. Задача обучения с подкреплением обычно включает подзадачу обучения тому, как функционирует среда, в которой существует агент.

В определении того, как должен действовать алгоритм обучения, важную роль играет также применяемое представление той информации, которая должна быть освоена в процессе обучения. Любой из компонентов агента можно представить с использованием любой из схем представления, описанных в этой книге. Выше уже было приведено несколько примеров: в программах ведения игр для представления функций полезности применяются полиномы с линейными весами; высказывания в пропозициональной логике и логике первого порядка применимы для представления всех компонентов логического агента; а вероятностные описания, такие как байесовские сети, применяются в компонентах агента, действующего на основе теории решений, которые предназначены для обеспечения вероятностного вывода. Для всех этих средств представления разработаны эффективные алгоритмы обучения. В данной главе рассматриваются методы, относящиеся к пропозициональной логике, в главе 19 описаны методы для логики первого порядка, а в главе 20 — методы для байесовских и нейронных сетей (которые включают линейные полиномы в качестве частного случая).

---

<sup>1</sup> Термин **подкрепление** является синонимом термина **вознаграждение**, который использовался в главе 17.

Последним важным аспектом проектирования обучающихся систем является наличие априорных знаний. Большинство исследований проблем обучения, проводимых в области искусственного интеллекта, компьютерных наук и психологии, касались того случая, в котором агент приступает к обучению, вообще не имея никаких знаний о том, что он пытается изучить. Он имеет доступ только к примерам, полученным на основе собственного опыта. Хотя этот случай представляет собой важный частный случай, его не следует рассматривать как общий случай. Основная часть человеческого обучения проходит в контексте, связанном с наличием большого объема фоновых знаний. Некоторые психологи и лингвисты утверждают, что даже новорожденные младенцы демонстрируют наличие у них знаний о мире. Но, независимо от того, являются ли эти утверждения истинными, нет никакого сомнения в том, что априорные знания могут оказаться огромной помощью в обучении. Физик, рассматривающий стопку фотографий, полученных с помощью пузырьковой камеры, может оказаться способным создать теорию, доказывающую существование новой частицы с определенной массой и зарядом, а если та же стопка фотографий будет показана без дополнительных пояснений искусствоведу, то он сможет лишь утверждать, будто этот “художник” относится к какой-то школе абстракционистов или экспрессионистов. В главе 19 будет показано несколько способов, позволяющих упростить обучение с использованием существующих знаний; кроме того, в ней показано, как можно компилировать знания для ускорения процесса принятия решений. В главе 20 описаны способы применения априорных знаний для упрощения изучения вероятностных теорий.

## 18.2. ИНДУКТИВНОЕ ОБУЧЕНИЕ

Любой алгоритм детерминированного контролируемого обучения получает в качестве исходной информации правильные значения неизвестной функции, соответствующие конкретным входным данным, и должен предпринять попытку восстановить эту неизвестную функцию или сформировать какую-то другую функцию, близкую к ней. Более формально можно определить, что **пример** представляет собой пару  $(x, f(x))$ , где  $x$  — входное, а  $f(x)$  — выходное значение функции, применяемой к  $x$ . Основная задача **чисто индуктивного логического вывода** (или просто **индукции**) указана ниже.

На основании совокупности примеров входных и выходных данных функции  $f$  получить функцию  $h$ , которая аппроксимирует  $f$ .

Функция  $h$  называется **гипотезой**. С концептуальной точки зрения та причина, по которой задача обучения является трудной, состоит в том, что обычно не легко определить, действительно ли какая-то конкретная функция  $h$  является хорошей аппроксимацией для  $f$ . Качественная гипотеза должна обеспечивать приемлемое **обобщение**, т.е. должна правильно предсказывать появление еще не полученных примеров. В этом состоит фундаментальная **проблема индукции**. Эта проблема изучалась в течение многих столетий; в разделе 18.5 приведено ее частичное решение.

На рис. 18.1 приведен известный пример: подгонка функции от одной переменной к некоторым точкам из набора данных. Примеры представляют собой

пары  $(x, f(x))$ , где и  $x$  и  $f(x)$  — действительные числа. Выберем в качестве пространства гипотез **Н** (множества гипотез, которые мы будем рассматривать в качестве потенциально приемлемых) множество полиномов, имеющих степень не больше  $k$ , таких как  $3x^2+2$ ,  $x^{17}-4x^3$  и т.д. На рис. 18.1, *a* показаны данные, которые соответствуют некоторой прямой (полиному первой степени). Эта прямая называется совместимой с гипотезой, поскольку она согласуется со всеми данными. На рис. 18.1, *b* показан полином более высокой степени, который также согласуется с этими данными. Данного случай может служить иллюстрацией к наиболее важной проблеме в индуктивном обучении: как осуществлять выбор среди многочисленных согласованных гипотез? Ответ состоит в использовании принципа бритвы Оккама<sup>2</sup>, согласно которому предпочтение следует отдавать наиболее простой гипотезе, согласующейся с данными. Интуитивно ясно, что такой подход имеет смысл, поскольку гипотезы не позволяют извлекать из данных какую-либо информацию, если они не проще самих данных. Определить, какая гипотеза проще, а какая сложнее, обычно нелегко, но, по-видимому, вполне резонным является утверждение, что полином первой степени проще по сравнению с полиномом двенадцатой степени.

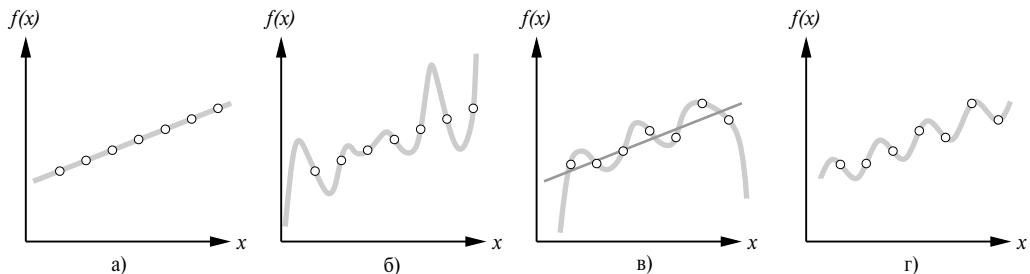


Рис. 18.1. Типичные задачи обучения: пример пар  $(x, f(x))$  и совместимой линейной гипотезы (*а*); совместимая гипотеза для того же набора данных в виде полинома седьмой степени (*б*); другой набор данных, который точно соответствует полиному шестой степени или приблизительно соответствует прямой линии (*в*); простая синусоидальная функция, которая точно соответствует тому же набору данных (*г*)

На рис. 18.1, *в* показан второй набор данных. С этим набором данных нельзя совместить прямую линию; в действительности для обеспечения точного согласования с ним требуется полином шестой степени (с семью параметрами). Количество точек равно только семи, поэтому полином должен иметь столько же параметров, сколько имеется точек данных; таким образом, создается впечатление, что этот полином не позволяет найти в данных какие-либо повторяющиеся шаблоны, и поэтому не следует ожидать, что с его помощью будет получено хорошее обобщение. Может оказаться, что лучше согласовать этот набор данных с прямой линией, которая не будет точно совместимой, но позволит получать вполне обоснованные предсказания. Принятие данного решения равносильно признанию такой возможности, что истинная функция не является детерминированной (или, что примерно эквивалентно

<sup>2</sup> Этот принцип получил название в честь английского философа XIV века Уильяма из Оккама, но в его обозначении название города, в котором работал философ, записывают как “Оккам”, в соответствии с французской транскрипцией, “Guillaume d'Occam” — Гийом из Оккама.

этому утверждению, истинные входные данные не являются полностью наблюдаемыми).  $\Leftrightarrow$  При наличии недетерминированных функций неизбежно приходится искать компромисс между сложностью гипотезы и степенью ее согласования с данными. В главе 20 показано, как достичь этого компромисса с помощью теории вероятностей.

Следует всегда учитывать, что возможность или невозможность найти простую, согласованную гипотезу зависит главным образом от выбранного пространства гипотез. На рис. 18.1, *г* показано, что данные, приведенные на рис. 18.1, *в*, могут быть точно согласованы с простой функцией в форме  $ax+b+c\sin x$ . Этот пример подчеркивает важность выбора пространства гипотез. Пространство гипотез, состоящее из полиномов конечной степени, не позволяет точно представить синусоидальные функции, поэтому ученик, использующий такое пространство гипотез, не сможет осуществить обучение с использованием синусоидальных данных. Принято считать, что задача обучения является  $\Leftrightarrow$  реализуемой, если пространство гипотез содержит подходящую функцию; в противном случае она является  $\Leftrightarrow$  нереализуемой. К сожалению, в любой ситуации невозможно сразу же определить, относится ли данная конкретная задача обучения к категории реализуемых, поскольку не известна истинная функция. Один из способов, позволяющих преодолеть этот барьер, состоит в использовании априорных знаний для логического вывода пространства гипотез, в котором, как известно, должна находиться истинная функция. Эта тема рассматривается более подробно в главе 19.

Еще один подход состоит в применении наибольшего возможного пространства гипотез. Например, почему бы не использовать в качестве  $h$  класс всех машин Тьюринга? В конечном итоге любая вычислимая функция может быть представлена с помощью некоторой машины Тьюринга, и это — лучший способ представления, который только может применяться. Но при реализации этой идеи возникает проблема, связанная с тем, что в ней не учитывается вычислительная сложность обучения.  $\Leftrightarrow$  Необходимо найти компромисс между выразительностью пространства гипотез и сложностью поиска простой, совместимой гипотезы в этом пространстве. Например, подгонка к данным прямых линий осуществляется очень просто; подбор полиномов высокой степени становится сложнее, а создание соответствующих машин Тьюринга действительно представляет собой очень сложную задачу, поскольку неразрешима в общем виде даже проблема определения того, является ли конкретная машина Тьюринга совместимой с данными. Второй причиной, по которой следует предпочесть простые пространства гипотез, является то, что результирующие гипотезы могут оказаться более простыми в использовании, т.е. вычисление  $h(x)$ , если  $h$  — линейная функция, будет осуществляться быстрее, чем при использовании программы, моделирующей произвольную машину Тьюринга.

По этим причинам основной объем исследований в области обучения был сосредоточен на относительно простых способах представления. В данной главе в основном рассматриваются пропозициональная логика и связанные с ней языки. В главе 19 рассматриваются теории обучения в логике первого порядка. Ниже будет показано, что компромисс между выразительностью и сложностью найти не так просто, как кажется на первый взгляд, поскольку, как было описано в главе 8, выразительный язык позволяет создать простую теорию, согласующуюся с данными, а ограничение выразительности языка приводит к тому, что любая согласованная теория должна стать очень сложной. Например, правила шахмат могут быть записаны в

логике первого порядка на одной или двух страницах, но потребуют тысячи страниц при их формулировке в пропозициональной логике. Кроме того, в подобных случаях при использовании более выразительного языка намного ускоряется обучение.

### 18.3. ФОРМИРОВАНИЕ ДЕРЕВЬЕВ РЕШЕНИЙ НА ОСНОВЕ ОБУЧЕНИЯ

---

Индуктивный логический вывод деревьев решений представляет собой одну из простейших и вместе с тем наиболее успешных форм алгоритмов обучения. Эта тема может служить хорошим введением в проблематику индуктивного обучения, а алгоритмы, разработанные в ее рамках, легко поддаются реализации. В настоящем разделе вначале описан производительный элемент проекта агента, а затем показано, как обеспечить его обучение. В ходе этого изложения представлены основные понятия, которые применяются во всех областях индуктивного обучения.

#### Деревья решений, рассматриваемые как производительные элементы

Дерево решений принимает в качестве входных данных объект или ситуацию, описанную с помощью множества атрибутов, и возвращает “решение” — предсказанное выходное значение, соответствующее входным данным. Входные атрибуты могут быть дискретными или непрерывными. На данный момент подразумевается, что входные данные являются дискретными. Выходные значения также могут быть дискретными или непрерывными; процесс формирования в ходе обучения функции с дискретными значениями называется обучением классификации; формирование в ходе обучения непрерывной функции называется обучением регрессии. Вначале мы сосредоточимся на булевой классификации, согласно которой каждый пример обозначается как истинный (положительный) или ложный (отрицательный).

Дерево решений позволяет перейти к содержащемуся в нем решению путем выполнения последовательности проверок. Каждый внутренний узел в дереве соответствует проверке значения одного из свойств, а ветви, исходящие из этого узла, обозначены возможными значениями результатов проверки. Каждый листовой узел в дереве задает значение, возвращаемое после достижения этого листа. Понятно, представление в виде дерева решений кажется людям вполне естественным; в действительности многие инструктивные руководства (например, по ремонту автомобилей) полностью оформлены в виде одного дерева решений, разбросанного по нескольким сотням страниц.

Несколько более простой пример может быть основан на применении методов обучения к задаче, в которой клиент ждет, пока освободится место за столиком в ресторане. Цель состоит в том, чтобы изучить определение для целевого предиката *WillWait* (Следует ли ждать). Подготавливая данный пример для использования в качестве задачи обучения, необходимо вначале определить, какие атрибуты доступны для описания примеров ситуаций в данной проблемной области. В главе 19 будет показано, как автоматизировать выполнение этого этапа, а на данный момент предположим, что решено использовать приведенный ниже список атрибутов.

1. *Alternate* (Альтернативный вариант). Есть ли поблизости подходящий ресторан такого же класса.
2. *Bar* (Бар). Имеется ли в ресторане уютный бар, в котором можно подождать.
3. *Fri/Sat* (Уик-энд). Принимает истинное значение по пятницам и субботам.
4. *Hungry* (Чувство голода). Испытывает ли посетитель чувство голода.
5. *Patrons* (Посетители). Сколько людей находится в ресторане; значениями этого атрибута являются *None* (Ресторан пуст), *Some* (В ресторане есть посетители) и *Full* (Ресторан заполнен).
6. *Price* (Цены). Ценовая категория ресторана (\*, \*\*, \*\*\*).
7. *Raining* (Дождь). Идет ли дождь на улице.
8. *Reservation* (Бронирование). Забронировано ли место за посетителем.
9. *Type* (Тип). Тип ресторана (ресторан с французской (*French*), итальянской (*Italian*), тайской (*Thai*) кухней или ресторан-закусочная (*Burger*)).
10. *WaitEstimate* (Оценка продолжительности ожидания). Продолжительность ожидания, оценка которой сделана метрдотелем (0–10, 10–30, 30–60, >60 минут).

Дерево решений, обычно используемое одним из авторов (Стюартом Расселом), для данной проблемной области, показано на рис. 18.2. Обратите внимание на то, что в этом дереве не используются атрибуты *Price* и *Type*; по сути это означает, что лицо, принимающее решение, рассматривает их как малозначимые. Обработка примеров ситуаций с помощью этого дерева начинается от корня и проходит по соответствующей ветви до тех пор, пока не будет достигнут какой-то лист. В частности, пример ситуации, в которой *Patrons=Full* и *WaitEstimate=0–10* будет рассматриваться как положительный (да, следует дождаться освобождения столика).

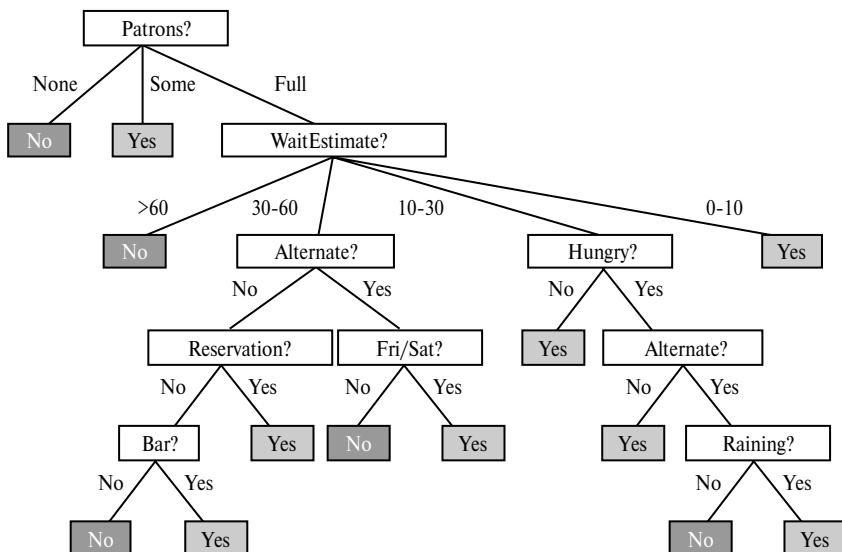


Рис. 18.2. Дерево решений для определения того, следует ли подождать, пока освободится столик

## Выразительность деревьев решений

С точки зрения логики любая конкретная гипотеза из дерева решений для целевого предиката *WillWait* может рассматриваться как утверждение в следующей форме:

$$\forall s \ WillWait(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \dots \vee P_n(s))$$

где каждое условие  $P_i(s)$  представляет собой конъюнкцию проверок, соответствующих пути от корня дерева к листу с положительным результатом. Хотя это выражение с виду напоминает высказывание в логике первого порядка, оно в определенном смысле является пропозициональным, поскольку содержит только одну переменную, а все предикаты являются унарными. Это дерево решений фактически описывает связь между предикатом *WillWait* и некоторой логической комбинацией значений атрибутов. Деревья решений не могут использоваться для представления проверок, относящихся к двум или нескольким разным объектам, например, таких проверок, как показано ниже (“Есть ли поблизости более дешевый ресторан?”).

$$\exists r_2 \ Nearby(r_2, r) \wedge Price(r, p) \wedge Price(r_2, p_2) \wedge Cheaper(p_2, p)$$

Очевидно, что можно было бы ввести еще один булев атрибут с именем *CheaperRestaurantNearby* (Наличие поблизости более дешевого ресторана), но задача введения всех подобных атрибутов является трудно осуществимой. В главе 19 приведены дополнительные сведения о задаче правильного обучения в логике первого порядка.

Деревья решений в рамках класса пропозициональных языков являются полностью выразительными; это означает, что в виде дерева решений может быть оформлена любая булева функция. Такую задачу можно выполнить тривиально, предложив, что каждая строка в истинностной таблице для данной функции соответствует определенному пути в дереве. Но подобный подход приводит к получению представления в виде дерева решений с экспоненциальными размерами, поскольку количество строк в истинностной таблице увеличивается экспоненциально, тогда как очевидно, что деревья решений позволяют представить многие функции с помощью гораздо меньших деревьев.

Но для некоторых видов функций такая проблема становится вполне реальной. Например, если рассматриваемая функция является **функцией определения четности**, которая возвращает 1 тогда и только тогда, когда на входах задано четное количество единиц, то потребуется дерево решений, имеющее экспоненциальную величину. Кроме того, сложной является задача использования дерева решений для представления **мажоритарной функции**, которая возвращает 1, если больше чем на половине ее входов имеется 1.

Иными словами, деревья решений вполне подходят для функций одних типов и не подходят для других. Существует ли представление какого-либо типа, являющееся эффективным для функций всех типов? К сожалению, ответ на этот вопрос отрицателен. Это утверждение можно доказать с помощью такого общего способа. Рассмотрим множество всех булевых функций от  $n$  атрибутов. Каково общее количество различных функций в этом множестве? Оно определяется количеством различных истинностных таблиц, которые могут быть составлены на этих атрибутах, поскольку функция определяется своей истинностной таблицей. Истинностная таблица имеет  $2^n$  строк, так как для описания вариантов входных данных применяется  $n$  атрибутов. Столбец “ответов” такой таблицы может рассматриваться как число, состоящее из

$2^n$  битов, которое определяет функцию. Независимо от того, какое представление используется для функций, некоторые из функций (а фактически почти все) должны потребовать для своего представления именно такое количество битов.

Если для определения функции требуется  $2^n$  битов, то существует  $2^{2^n}$  различных функций от  $n$  атрибутов. Это число является весьма обескураживающим. Например, при наличии лишь шести булевых атрибутов существует  $2^{2^6} = 18\ 446\ 744\ 073\ 709\ 551\ 616$  различных функций, из числа которых может быть сделан выбор. Для поиска совместимых гипотез в таком большом пространстве потребуется некоторые остроумные алгоритмы.

### Индуктивный вывод деревьев решений на основе примеров

Пример для булева дерева решений состоит из вектора входных атрибутов  $X$  и одного булева выходного значения  $y$ . Множество примеров  $(X_1, y_1), \dots, (X_{12}, y_{12})$  показано в табл. 18.1. Положительными примерами являются те, в которых цель *WillWait* имеет истинное значение  $(X_1, X_3, \dots)$ , а отрицательными — те, в которых эта цель имеет ложное значение  $(X_2, X_5, \dots)$ . Полное множество примеров называется **обучающим множеством**.

Таблица 18.1. Примеры для проблемной области задачи с рестораном

Пример	Атрибуты										Цель
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
$X_1$	Yes	No	No	Yes	Some ***	No	Yes	French	0–10	Yes	
$X_2$	Yes	No	No	Yes	Full *	No	No	Thai	30–60	No	
$X_3$	No	Yes	No	No	Some *	No	No	Burger	0–10	Yes	
$X_4$	Yes	No	Yes	Yes	Full *	Yes	No	Thai	10–30	Yes	
$X_5$	Yes	No	Yes	No	Full ***	No	Yes	French	760	No	
$X_6$	No	Yes	No	Yes	Some **	Yes	Yes	Italian	0–10	Yes	
$X_7$	No	Yes	No	No	None *	Yes	No	Burger	0–10	No	
$X_8$	No	No	No	Yes	Some **	Yes	Yes	Thai	0–10	Yes	
$X_9$	No	Yes	Yes	No	Full *	Yes	No	Burger	760	No	
$X_{10}$	Yes	Yes	Yes	Yes	Full ***	No	Yes	Italian	10–30	No	
$X_{11}$	No	No	No	No	None *	No	No	Thai	0–10	No	
$X_{12}$	Yes	Yes	Yes	Yes	Full *	No	No	Burger	30–60	Yes	

Проблема поиска дерева решений, которое согласуется с обучающим множеством, на первый взгляд может показаться сложной, но фактически имеет простейшее решение. Можно просто сформировать дерево решений, в котором имеется по одному пути к листовому узлу для каждого примера, где в пути проверяется каждый атрибут по очереди и происходит переход к значению для данного примера, а листовой узел содержит классификацию для данного примера. После повторного поступления того же примера<sup>3</sup> дерево решений обеспечивает правильную классификацию. Но, к сожалению, оно не позволяет учить какие-либо иные случаи!

<sup>3</sup> При этом нужно различать тот же пример или пример с тем же описанием; это различие очень важно, и мы вернемся к нему в главе 19.

Недостатком такого простейшего дерева решений является то, что в нем просто запоминаются результаты наблюдений. Оно не позволяет извлечь какие-либо закономерности из примеров, поэтому нельзя надеяться на то, что оно даст возможность экстраполировать примеры, которые еще не встречались. Применяя принцип бритвы Оккама, необходимо вместо этого найти наименьшее дерево решений, совместимое с рассматриваемым примером. К сожалению, применительно к любому обоснованному определению понятия “наименьший” задача поиска наименьшего дерева является трудноразрешимой. Однако с помощью некоторой простой эвристики можно многое достичь в направлении поиска “меньшего” дерева. Основная идея, лежащая в основе алгоритма Decision-Tree-Learning, состоит в том, что в первую очередь следует проверять наиболее важный атрибут. Под “наиболее важным” атрибутом подразумевается такой атрибут, который вносит наибольшее различие в классификацию примера. Таким образом, можно рассчитывать на обеспечение правильной классификации с помощью небольшого количества проверок, а это означает, что все пути в дереве будут короткими, а само дерево в целом окажется небольшим.

На рис. 18.3 показано, как начинается работа алгоритма. Ему предъявляются 12 обучающих примеров, которые классифицированы на множества положительных и отрицательных примеров. После этого принимается решение о том, какой атрибут должен использоваться для первой проверки дерева. На рис. 18.3, *a* показано, что *Type* — неподходящий атрибут, поскольку после его проверки остаются четыре возможных результата, каждый из которых содержит одинаковое количество положительных и отрицательных примеров. С другой стороны, на рис. 18.3, *b* можно видеть, что *Patrons* — довольно важный атрибут, поскольку если его значение равно *None* или *Some*, то остаются такие множества примеров, для которых можно сразу же получить определенный ответ (*No* и *Yes* соответственно). А если значением является *Full*, то остается смешанное множество примеров. Вообще говоря, после того, как проверка первого атрибута разбивает множество примеров на подмножества, каждый результат сам становится определением новой задачи обучения дерева решений с меньшим количеством примеров и количеством атрибутов, сократившимся на единицу. Ниже описаны четыре случая, которые следует рассматривать при анализе подобных рекурсивных задач.

1. Если имеются некоторые положительные и отрицательные примеры, то следует выбрать наилучший атрибут для выполнения по нему разбиения. Как показано на рис. 18.3, *b*, для разбиения оставшихся примеров используется атрибут *Hungry*.
2. Если все оставшиеся примеры являются положительными (или отрицательными), то работа закончена и можно дать ответ *Yes* или *No*. На рис. 18.3, *b* показаны примеры достижения этой цели в случаях *None* и *Some*.
3. Если не осталось ни одного примера, это означает, что соответствующие примеры не наблюдались, поэтому следует возвратить применяемое по умолчанию значение, вычисленное на основании мажоритарной классификации в родительском узле данного узла.
4. Если не осталось ни одного атрибута, но имеются и положительные и отрицательные примеры, то налицо проблема. Такая ситуация означает, что данные примеры имеют полностью одинаковое описание, но классифицированы по-

разному. Такое происходит, если некоторые данные определены неправильно; принято говорить, что в данных присутствует шум. Кроме того, такое происходит, либо если атрибуты не предоставляют достаточной информации для полного описания ситуации, либо если данная проблемная область действительно является недетерминированной. Одним из простых способов преодоления такой проблемы является использование мажоритарного голосования.

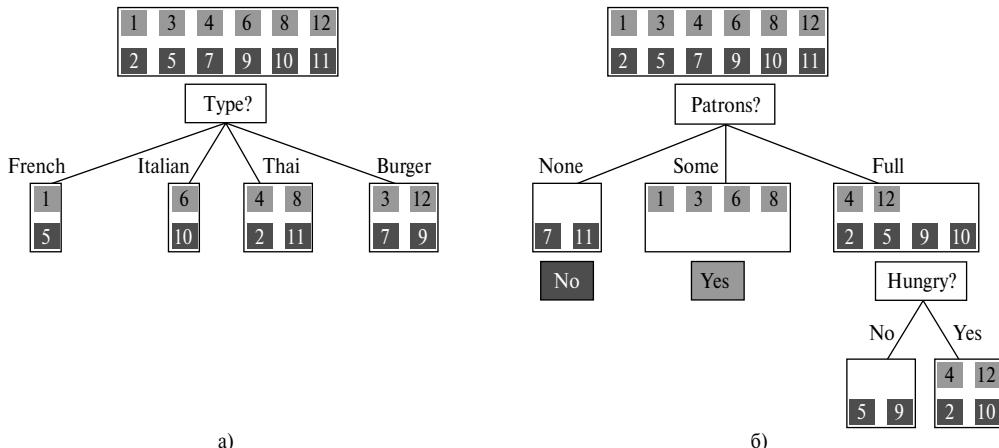


Рис. 18.3. Разбиение множества примеров на подмножества путем проверки атрибутов: в результате разбиения по атрибуту Type не удается приблизиться к решению задачи проведения различий между положительными и отрицательными примерами (а); разбиение по атрибуту Patrons позволяет многое добиться в части разделения положительных и отрицательных примеров (б). После разбиения по атрибуту Patrons довольно неплохой второй проверкой становится проверка по атрибуту Hungry

Алгоритм Decision-Tree-Learning показан в листинге 18.1. Подробные сведения о методе Choose-Attribute приведены в следующем подразделе.

#### Листинг 18.1. Алгоритм обучения дерева решений

```

function Decision-Tree-Learning(examples, attrs, default) returns дерево
  решений
  inputs: examples, множество примеров
           attrs, множество атрибутов
           default, заданное по умолчанию значение для целевого
           предиката

  if множество examples пусто then return default
  else if все примеры имеют одну и ту же классификацию
    then return классификация
  else if множество attrs пусто then return Majority-Value(examples)
  else
    best ← Choose-Attribute(attrs, examples)
    tree ← новое дерево решений с результатом best проверки
           корневого узла
    m ← Majority-Value(examples)
    for each значение vi результата best do
      tree[best] ← new Node(vi)
      tree[best].examples ← examples
      tree[best].value ← vi
      tree[best].count ← 1
      examples ← remove(examples, vi)
      tree[best].children ← Decision-Tree-Learning(examples, attrs, vi)
  
```

```

examplesi ← {элементы множества examples со
значениями best=vi}
subtree ← Decision-Tree-Learning(examplesi, attribs-best, m)
добавить к дереву решений tree ветвь с меткой vi и
поддеревом subtree
return tree

```

Окончательное дерево решений, полученное с помощью этого алгоритма после применения к набору данных с 12 примерами, показано на рис. 18.4. Очевидно, что указанное дерево отличается от исходного дерева, показанного на рис. 18.2, несмотря на тот факт, что обучающие данные в действительности были сформированы агентом, использовавшим исходное дерево. Можно было бы прийти к заключению, что этот обучающий алгоритм не очень хорошо справляется с задачей изучения правильной функции. Однако такое заключение далеко от истины. В обучающем алгоритме рассматриваются примеры, а не правильная функция, и на самом деле сформированная в нем гипотеза (см. рис. 18.4) не только согласуется со всеми примерами, но и значительно проще по сравнению с первоначальным деревом. В процессе работы обучающего алгоритма не было обнаружено причин для включения проверок значений атрибутов *Raining* и *Reservation*, поскольку алгоритм мог классифицировать все примеры без них. Кроме того, алгоритм обнаружил интересный и ранее не наблюдавшийся шаблон поведения: первый автор, указанный на обложке данной книги, готов ждать, чтобы поесть тайской пищи по уик-эндам.

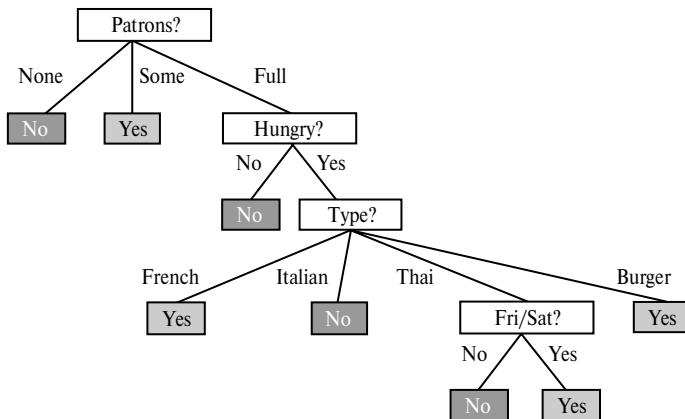


Рис. 18.4. Дерево решений, сформированное на основании обучающего множества с 12 примерами

Безусловно, если бы было собрано больше примеров, то вполне возможно, что сформированное дерево в большей степени напоминало бы исходное. К тому же дерево, приведенное на рис. 18.4, способно выработать ошибочное решение, например, при его формировании никогда не встречался случай, в котором требовалось ждать от 0 до 10 минут, а ресторан был полон. На ту ситуацию, в которой атрибут *Hungry* имеет ложное значение, дерево сообщает, что ждать не следует, тогда как сам автор (Стюарт Рассел), безусловно, стал бы ждать. Исходя из этого, возникает очевидный вопрос: если алгоритм выводит на основании примеров согласованное, но неправильное дерево, то насколько неправильным может вообще оказаться это

дерево? Ниже, после рассмотрения подробных сведений об этапе выбора атрибутов, будет показано, как проводить экспериментальный анализ этого вопроса.

## Выбор проверок атрибутов

Подход к выбору атрибутов, используемый в обучении дерева решений, предназначен для минимизации глубины окончательно сформированного дерева. Идея этого подхода состоит в том, что следует выбирать в первую очередь такой атрибут, который позволяет сразу же выполнить максимально возможный объем работы по обеспечению правильной классификации примеров. Идеальный атрибут позволяет разделить множество примеров на подмножества, целиком состоящие из положительных или отрицательных примеров. Атрибут *Patrons* не идеален, но является достаточно приемлемым. С другой стороны, действительно бесполезный атрибут, такой как *Tуре*, создает подмножества примеров приблизительно с такими же соотношениями положительных и отрицательных примеров, как и в первоначальном множестве.

Таким образом, нам достаточно лишь определить формальный критерий оценки понятий “достаточно приемлемый” и “действительно бесполезный”, после чего появится возможность реализовать функцию *Choose-Attribute*, применяемую в алгоритме, который приведен в листинге 18.1. Этот критерий должен принимать максимальное значение, когда атрибут является идеальным, и минимальное значение, когда атрибут полностью бесполезен. Одним из подходящих критериев является ожидаемый объем ~~и~~ информации, предоставляемый атрибутом; в данном определении термин “информация” используется в математическом смысле, впервые определенном Шенноном и Увером [1394]. Чтобы понять суть концепции информации, достаточно представить себе, что она является ответом на вопрос, например, о том, упадет ли подброшенная монета орлом вверх. Объем информации, содержащийся в ответе на этот вопрос, зависит от наших априорных знаний. Чем меньше мы знаем, тем больше получим информации. В теории информации информационное содержание ответа измеряется в битах. Один бит информации является достаточным для ответа “да” или “нет” на вопрос о том, что полностью неизвестно, например, каков результат подбрасывания подлинной монеты. Вообще говоря, если возможные ответы  $v_i$  имеют вероятности  $P(v_i)$ , то информационное содержание  $I$  фактического ответа определяется с помощью следующего уравнения:

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$

Для проверки этого уравнения рассмотрим случай с подбрасыванием подлинной монеты. При этом будет получено следующее:

$$I\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ бит}$$

А если центр тяжести монеты смешен таким образом, что в 99% выпадает орел, то будет получено  $I(1/100, 99/100) = 0.08$  битов, а поскольку вероятность выпадения орла приближается к 1, информационное содержание фактического ответа приближается к 0.

Что касается обучения деревьев решений, то необходимо найти ответ на вопрос — какова правильная классификация для данного примера? Правильное дерево решений должно позволить найти ответ на этот вопрос. Оценка вероятностей возможных ответов, полученная перед тем, как будет выполнена проверка какого-либо из этих атрибутов, определяется соотношениями положительных и отрицательных примеров в обучающем множестве. Предположим, что обучающее множество включает  $p$  положительных примеров и  $n$  отрицательных. В таком случае оценка информации, содержащейся в правильном ответе, равна:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Обучающее множество для задачи с рестораном, приведенное в табл. 18.1, включает количество примеров  $p=n=6$ , поэтому требуется 1 бит информации.

Теперь отметим, что обычно проверка по одному атрибуту  $A$  не позволит нам получать такой объем информации, но по меньшей мере предоставит часть этой информации. Мы можем точно измерить, какова эта часть, определяя, сколько еще информации нам потребуется после проверки этого атрибута. Любой атрибут  $A$  делит обучающее множество  $E$  на подмножества  $E_1, \dots, E_v$  в соответствии со значениями атрибута  $A$ , если  $A$  может иметь  $v$  различных значений. Каждое подмножество  $E_i$  включает  $p_i$  положительных примеров и  $n_i$  отрицательных примеров, поэтому после перехода по ветви этого атрибута нам потребуется дополнительно  $I(p_i/(p_i+n_i), n_i/(p_i+n_i))$  битов информации, чтобы ответить на вопрос. Случайно выбранный пример из обучающего множества содержит  $i$ -е значение рассматриваемого атрибута с вероятностью  $(p_i+n_i)/(p+n)$ , поэтому в среднем после проверки атрибута  $A$  для классификации данного примера потребуется показанное ниже количество бит информации.

$$\text{Remainder}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

**Приращение информации**, полученное в результате проверки атрибута, представляет собой разность между первоначальной потребностью в информации и новой потребностью:

$$\text{Gain}(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Remainder}(A)$$

Эвристика, используемая в функции `Choose-Attribute`, состоит в том, что следует выбирать атрибут с наибольшим приращением. Возвращаясь к атрибутам, показанным на рис. 18.3, получаем следующее:

$$\text{Gain}(\text{Patrons}) = 1 - \left[ \frac{2}{12} I(0, 1) + \frac{4}{12} I(1, 0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] \approx 0.541 \text{ бит}$$

$$\text{Gain}(\text{Type}) = 1 - \left[ \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0$$

Эти соотношения подтверждают интуитивное предположение о том, что *Patrons* — наилучший атрибут, по которому следует выполнять разбиение. В действительности атрибут *Patrons* позволяет получить наибольшее приращение информации по сравнению с любыми другими атрибутами и должен быть выбран алгоритмом обучения дерева решений в качестве корневого.

### Оценка производительности обучающего алгоритма

Обучающий алгоритм является приемлемым, если он вырабатывает гипотезы, обеспечивающие качественные предсказания в отношении того, к какому классу будут относиться еще не встречавшиеся примеры. В разделе 18.5 будет показано, как можно заранее оценить качество предсказания. А в данном разделе рассматривается методология оценки качества предсказания после его фактического получения.

Очевидно, что предсказание является качественным, если оно — истинное, поэтому можно оценить качество гипотезы, сверяя ее предсказания с правильной классификацией после того, как она становится известной. Такая задача осуществляется с помощью множества примеров, которые принято называть **проверочным множеством**. Дело в том, что если обучение проведено по всем доступным примерам, то приходится возвращаться к действительности и заниматься подбором дополнительных примеров для проведения по ним проверки, поэтому чаще более удобно использовать описанную ниже методологию.

1. Собрать множество примеров большого объема.
2. Разделить его на два непересекающихся подмножества: **обучающее множество** и **проверочное множество**.
3. Применить обучающий алгоритм к обучающему множеству для формирования гипотезы  $h$ .
4. Определить, какой процент примеров в проверочном множестве правильно классифицируется с помощью гипотезы  $h$ .
5. Повторять этапы 2–4 для различных размеров обучающих множеств и различных случайно выбранных обучающих множеств каждого размера.

Результатом применения этой процедуры становится набор данных, которые можно обрабатывать для получения среднего качества предсказаний, которое выражается в виде некоторой функциональной зависимости от размера обучающего множества. Эта функция может быть изображена в виде графика, представляющего собой так называемую **кривую обучения** для данного алгоритма в данной конкретной проблемной области. Кривая обучения для алгоритма Decision-Tree-Learning, полученная с использованием примеров из задачи с рестораном, показана на рис. 18.5. Обратите внимание на то, что качество предсказания повышается по мере увеличения размера обучающего множества (по этой причине подобные кривые часто называют **счастливыми графиками**). Это — хороший признак, который показывает, что в данных действительно есть какие-то повторяющиеся шаблоны (закономерности) и обучающий алгоритм их выделяет.

Безусловно, что обучающему алгоритму не должно быть разрешено “касаться” проверочных данных перед тем, как по ним будет проверена изученная гипотеза. К сожалению, часто очень легко можно попасть в ловушку, связанную с тем, что

алгоритм ~~как~~ компрометирует проверочные данные. Компрометация обычно происходит следующим образом: в обучающем алгоритме могут быть предусмотрены все возможные “регуляторы”, предназначенные для настройки его поведения, например различные и разнотипные критерии выбора следующего атрибута в процессе обучения дерева решений. Итак, формируются гипотезы для всевозможных различных установок этих регуляторов, измеряется их производительность на проверочном множестве и формируется отчет о производительности предсказания наилучшей гипотезы. К сожалению, при этом происходит компрометация! Причина этого состоит в том, что гипотеза была выбрана по результатам измерения ее производительности на проверочном множестве, поэтому информация о проверочном множестве проникла в обучающий алгоритм. Мораль этой истории состоит в том, что в любом процессе, предусматривающем сравнение производительностей гипотез на проверочном множестве, необходимо использовать новое проверочное множество для измерения производительности окончательно выбранной гипотезы. Но на практике такой подход осуществлять слишком сложно, поэтому исследователи по-прежнему продолжают выполнять эксперименты на бывших в употреблении множествах примеров.



Рис. 18.5. Кривая обучения для алгоритма обучения дерева решений на 100 случайно сформированных примерах в проблемной области задачи с рестораном. В этом графике подытожены результаты 20 попыток

### Шум и чрезмерно тщательная подгонка

Как было показано выше, если имеются два или несколько примеров с одинаковым описанием (с точки зрения атрибутов), но с разными классификациями, то работа алгоритма Decision-Tree-Learning обязательно окончится неудачей, поскольку невозможно будет найти дерево решений, совместимое со всеми примерами. Кроме того, уже упоминалось, что приемлемый способ решения этой проблемы может предусматривать либо применение в каждом листовом узле мажоритарной классификации для относящегося к нему множества примеров (если требуется детерминированная гипотеза), либо формирование оценок вероятностей каждой классификации с использованием относительных частот. К сожалению, описанные выше ситуации не исчерпывают перечень всех возможных нарушений в процессе фор-

мирования дерева решений. Вполне возможна такая ситуация (которая действительно часто встречается на практике), что алгоритм обучения деревьев решений формирует дерево решений, совместимое со всеми примерами, даже несмотря на то, что эти примеры не содержат крайне важной информации для данной задачи классификации. Это связано с тем, что в рассматриваемом алгоритме могут использоваться не относящиеся к делу атрибуты (если они имеются), в результате чего проводятся несуществующие различия между примерами.

Рассмотрим задачу, в которой осуществляются попытки предсказать результаты броска игральной кости. Предположим, что эксперименты проводятся в течение продолжительного периода времени с различными игральными костями и что атрибуты, описывающие каждый обучающий пример, являются следующими:

1. *Day* (День недели). День, в который был выполнен бросок игральной кости (*Mon* (Понедельник), *Tue* (Вторник), *Wed* (Среда), *Thu* (Четверг)).
2. *Month* (Месяц). Месяц, в который был выполнен бросок игральной кости (*Jan* (Январь) или *Feb* (Февраль)).
3. *Color* (Цвет). Цвет игральной кости (*Red* (Красный) или *Blue* (Синий)).

При условии, что не существует двух примеров с одинаковыми описаниями и разными классификациями, алгоритм Decision-Tree-Learning позволяет найти точную гипотезу. Чем больше количество используемых атрибутов, тем выше вероятность, что будет найдена точная гипотеза. Но все такие гипотезы будут полностью не связанными с действительностью, поскольку рассматриваемые атрибуты не влияют на выпадение игральной кости. Требуется лишь то, чтобы алгоритм Decision-Tree-Learning возвратил единственный листовой узел с вероятностями, близкими к 1/6, для каждого результата выпадения очков на игральной кости, как только будет получено достаточное количество примеров.

Каждый раз, когда приходится сталкиваться с множеством возможных гипотез, имеющим большой объем, необходимо тщательно следить за тем, чтобы возникающая при этом свобода выбора не использовалась для поиска бессмысленных “закономерностей” в данных. Эта проблема называется **чрезмерно тщательной подгонкой**. На практике очень часто встречается такой феномен, что чрезмерно тщательная подгонка происходит, даже если целевая функция вообще не является случайной. Указанный недостаток возникает в обучающих алгоритмах любого типа, а не только в алгоритмах обучения деревьев решений.

Полная математическая трактовка проблемы чрезмерно тщательной подгонки выходит за рамки данной книги. Но в этом разделе представлен простой метод, называемый **отсечением ветвей дерева решений**, позволяющий в определенной степени справиться с указанной проблемой. Метод отсечения ветвей действует по принципу предотвращения рекурсивного разбиения по атрибутам, релевантность которых не является очевидной, даже если в соответствующем узле дерева имеются данные, не классифицированные на подмножества с равным количеством положительных и отрицательных примеров. Вопрос состоит в том, как обнаружить нерелевантный атрибут.

Предположим, что осуществляется разбиение множества примеров с использованием какого-то нерелевантного атрибута. Вообще говоря, следует полагать, что результирующие подмножества в этом случае будут иметь приблизительно такие же

соотношения количества примеров из каждого класса, как и первоначальное множество. Это означает, что приращение информации будет близким к нулю<sup>4</sup>. Таким образом, хорошим показателем релевантности атрибута является приращение информации. В таком случае возникает вопрос, насколько большим должно быть это приращение для того, чтобы имело смысл осуществлять разбиение по какому-то конкретному атрибуту?

На этот вопрос можно ответить с использованием статистической проверки значимости. Такая проверка начинается с принятия предположения о том, что в данных нет никаких скрытых закономерностей (это предположение называется нуль-гипотезой). После этого проводится анализ фактических данных для определения того, в какой степени они отличаются от данных, характеризующихся абсолютным отсутствием закономерностей. Если полученная степень отклонения является статистически маловероятной (обычно принято считать, что вероятность отклонения составляет 5% или меньше), то такие данные рассматриваются как надежное свидетельство наличия значимых закономерностей в данных. Вероятности вычисляются на основании стандартных распределений величины отклонения, которые можно надеяться обнаружить в случайно сформированных выборках.

В данном случае нуль-гипотеза состоит в том, что атрибут является нерелевантным, а следовательно, приращение информации для бесконечно большого образца будет равно нулю. Необходимо вычислить вероятность того, что после принятия нуль-гипотезы образец с размером  $v$  будет показывать наблюдаемое отклонение от ожидаемого распределения положительных и отрицательных примеров. Такое отклонение можно измерить, сравнивая фактические количества положительных и отрицательных примеров в каждом подмножестве,  $p_i$  и  $n_i$ , с ожидаемыми количествами,  $\hat{p}_i$  и  $\hat{n}_i$ , при том условии, что принято предположение об истинной нерелевантности атрибута:

$$\hat{p}_i = p \times \frac{p_i + n_i}{p+n} \quad \hat{n}_i = n \times \frac{p_i + n_i}{p+n}$$

Удобный критерий суммарного отклонения определяется следующей формулой:

$$D = \sum_{i=1}^v \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{\hat{n}_i}$$

Согласно нуль-гипотезе, значение  $D$  распределяется в соответствии с распределением  $\chi^2$  (хи-квадрат) с  $v-1$  степенями свободы. Вероятность того, что атрибут действительно является нерелевантным, можно рассчитать с помощью стандартных таблиц  $\chi^2$  или с применением статистического программного обеспечения. В упр. 18.11 предлагается внести соответствующие изменения в алгоритм Decision-Tree-Learning для реализации этой формы отсечения, которая известна под названием отсечение  $\chi^2$ .

Отсечение позволяет также справиться с шумом: ошибки классификации приводят к линейному увеличению ошибки предсказания, а ошибки в описаниях примеров оказывают асимптотическое влияние, которое становится все более ярко выражено.

<sup>4</sup> В действительности это приращение всегда будет положительным, за исключением тех случаев, когда все эти соотношения будут абсолютно одинаковыми (см. упр. 18.10).

женным по мере того, как дерево все больше сокращается до меньших множеств. Деревья, сформированные с помощью отсечения, показывают гораздо лучшую производительность, чем деревья, сформированные без отсечения, в том случае, если данные содержат большой объем шума. Деревья с отсеченными ветвями часто бывают намного меньше, поэтому их структуру проще понять.

Еще одним методом, позволяющим уменьшить влияние чрезмерно тщательной подгонки, является **перекрестная проверка**. Этот метод применим к любому алгоритму обучения, а не только к алгоритму обучения деревьев решений. Основная идея этого метода состоит в том, что можно оценить, насколько хорошо каждая гипотеза строит предсказание по данным, не встречавшимся ранее. Этот метод осуществляется на практике путем резервирования определенной части известных данных и дальнейшего использования их для проверки производительности предсказания той гипотезы, которая выведена из оставшихся данных. В методе  $k$ -кратной перекрестной проверки предусматривается проведение  $k$  экспериментов с резервированием каждый раз другой части данных для проведения проверки, объем которой равен  $1/k$  от первоначальных данных, с последующим усреднением результатов. В качестве значения  $k$  часто используются 5 и 10. В пределе в данном методе может применяться  $k=n$ ; такой вариант метода называют также перекрестной проверкой с исключением одного примера (*leave-one-out cross-validation*). Перекрестная проверка может использоваться в сочетании с любым методом формирования дерева (включая метод отсечения ветвей) в целях создания дерева, характеризующегося высокой производительностью предсказания. В дальнейшем, чтобы исключить возможность компрометации проверочных данных, необходимо провести измерение производительности полученного дерева с помощью нового проверочного множества.

## Расширение области применения деревьев решений

Для того чтобы распространить методы индуктивного вывода деревьев решений на более широкий круг задач, необходимо решить целый ряд проблем. В данном разделе кратко описана каждая из этих проблем, но более полного их понимания можно добиться, выполнив указанные здесь упражнения.

- **Недостающие данные.** Во многих проблемных областях не все значения атрибутов могут быть определены для каждого примера, в связи с тем, что такие значения могут оказаться незарегистрированными или задача их получения является слишком дорогостоящей. Такая ситуация приводит к возникновению двух проблем. Во-первых, если данное дерево решений, то как следует классифицировать некоторый объект, для которого не задан один из проверяемых атрибутов? Во-вторых, как следует модифицировать формулу приращения информации, если в некоторых примерах неизвестны значения данного атрибута? Эти вопросы рассматриваются в упр. 18.12.
- **Многозначные атрибуты.** Если атрибут имеет много возможных значений, то критерий приращения информации придает оценке полезности атрибута не соответствующую ей значимость. В крайнем случае может встретиться такой атрибут, который имеет в каждом примере другое значение, скажем *RestaurantName* (Название ресторана). В таком случае каждое подмножество примеров становится одноэлементным подмножеством с уникальной клас-

сификацией, поэтому критерий приращения информации для соответствующего атрибута принимает наивысшее значение. Тем не менее этот атрибут может оказаться нерелевантным или бесполезным. Одним из решений данной проблемы является использование **коэффициента приращения** (упр. 18.13).

- **Непрерывные и целочисленные входные атрибуты.** Непрерывные или целочисленные атрибуты, такие как *Height* (Рост) и *Weight* (Вес), имеют бесконечное множество возможных значений. Но вместо формирования бесконечно большого количества ветвей алгоритмы обучения деревьев решений, как правило, находят **точку разбиения**, позволяющую получить наивысшее приращение информации. Например, в каком-то конкретном узле дерева может оказаться, что наибольший объем информации позволяет получить проверка по условию  $Weight > 160$ . Разработаны эффективные методы динамического программирования для поиска приемлемых точек разбиения, но они все еще представляют собой тот компонент реальных приложений в области обучения деревьев решений, который требует намного больше затрат по сравнению с другими компонентами.
- **Выходные атрибуты с непрерывными значениями.** Если предпринимается попытка предсказать некоторое числовое значение, такое как оценка произведения искусства, а не провести дискретную классификацию, то необходимо получить **дерево регрессии**. В каждом листовом узле такого дерева задана линейная функция от некоторого подмножества числовых атрибутов, а не единственное значение. Например, ветвь, которая относится к гравюрам, раскрашенным вручную, может оканчиваться линейной функцией от площади, возраста работы и количества цветов. Обучающий алгоритм должен выработать решение о том, когда следует прекратить разбиение и приступить к применению метода линейной регрессии с использованием оставшихся атрибутов (или некоторого их подмножества).

Система обучения деревьев решений для реальных приложений должна быть способной решать все эти проблемы. Особенно важной является обработка переменных с непрерывными числовыми значениями, поскольку числовые данные применяются, например, в физических и финансовых процессах. Было разработано несколько коммерческих пакетов, соответствующих этим критериям, которые использовались для создания нескольких сотен проблемно-ориентированных систем. Во многих областях промышленности и торговли деревья решения становятся первым методом, к которому пытаются прибегнуть, когда из некоторого набора данных необходимо извлечь соответствующий ему метод классификации. Одним из важных свойств деревьев решений является то, что вывод обучающего алгоритма доступен для понимания людей (а в действительности в этом также состоит законодательное требование к финансовым решениям, на которые распространяются законы против дискриминации). Таким свойством не обладают нейронные сети (см. главу 20).

## 18.4. ОБУЧЕНИЕ АНСАМБЛЯ

До сих пор в этой главе рассматривались методы обучения, в которых для получения предсказаний использовалась отдельная гипотеза, выбранная из пространства гипотез. В отличие от этого, идея методов **обучения ансамбля** состоит в том, что из

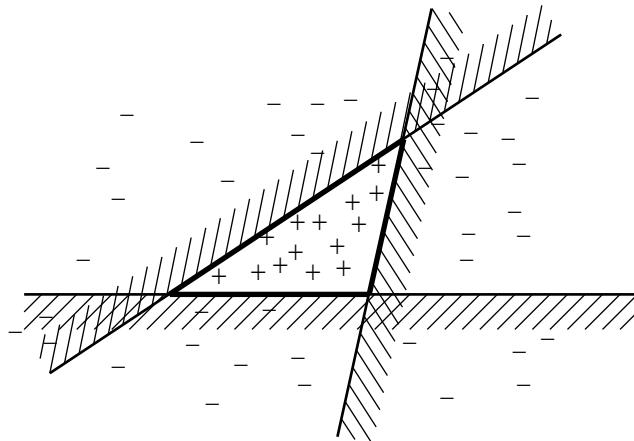
пространства гипотез следует выбрать целую коллекцию, или так называемый **ансамбль** гипотез, и в дальнейшем комбинировать предсказания, полученные с помощью гипотез этого ансамбля. Например, может быть сформировано сто разных деревьев решений из одного и того же обучающего множества, после чего проведено голосование для определения наилучшей классификации нового примера.

В основе стремления использовать обучение ансамбля лежит простая причина. Рассмотрим ансамбль из  $M=5$  гипотез и предположим, что их предсказания комбинируются с использованием несложного мажоритарного голосования. Для того чтобы этот ансамбль неправильно классифицировал новый пример, его должны неправильно классифицировать по меньшей мере три из пяти гипотез. Однако вполне можно рассчитывать на то, что данная ситуация является гораздо менее вероятной по сравнению с ошибочной классификацией при использовании единственной гипотезы. Допустим, что предполагается, будто каждая гипотеза  $h_i$  в ансамбле допускает ошибку с вероятностью  $p$ . Иными словами, вероятность того, что случайно выбранный пример будет неправильно классифицирован гипотезой  $h_i$ , равна  $p$ . Кроме того, допустим, что предполагается, будто ошибки, допущенные с применением каждой гипотезы, являются независимыми. В таком случае, если вероятность  $p$  мала, то вероятность одновременного появления большого количества ошибок классификации становится микроскопической. Например, простой расчет (упр. 18.14) показывает, что использование ансамбля из пяти гипотез позволяет сократить частоту ошибок от величины  $1/10$  до величины меньше чем  $1/100$ . Тем не менее очевидно, что предположение о независимости гипотез неоправданно, поскольку во всех гипотезах, скорее всего, будут возникать одинаковые искажения, вызванные одними и теми же искажающими их аспектами одинаковых обучающих данных. Но если гипотезы хоть немного отличаются друг от друга, что приводит к уменьшению корреляции между их ошибками, то обучение ансамбля может оказаться очень полезным.

Еще один способ трактовки идеи ансамбля состоит в том, что ансамбль — это универсальный метод расширения пространства гипотез. Это означает, что сам ансамбль может рассматриваться как гипотеза, а новое пространство гипотез — как множество всех возможных ансамблей, которые могут быть сформированы из гипотез первоначального пространства. Как показано на рис. 18.6, такой подход может привести к созданию более выразительного пространства гипотез. Если первоначальное пространство гипотез допускает возможность использовать простой и эффективный алгоритм обучения, то метод формирования ансамбля предоставляет возможность формировать в процессе обучения гораздо более выразительный класс гипотез без значительного дополнительного увеличения вычислительной или алгоритмической сложности.

Наиболее широко используемый метод формирования ансамбля называется **усищением**. Для того чтобы понять, как он работает, необходимо вначале ознакомиться с идеей **взвешенного обучающего множества**. В таком обучающем множестве с каждым примером связан вес  $w_j \geq 0$ . Чем больше вес примера, тем выше важность, присвоенная ему в процессе изучения какой-то гипотезы. Рассматриваемые до сих пор в этой главе алгоритмы обучения несложно модифицировать для работы со взвешенными обучающими множествами<sup>5</sup>.

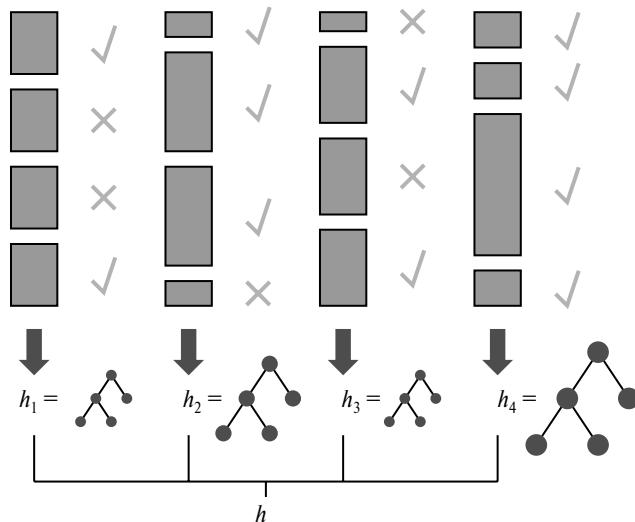
<sup>5</sup> Имеются также такие обучающие алгоритмы, которые не предоставляют подобной возможности. Но для них вместо этого можно создать **тиражированное обучающее множество**, в котором  $i$ -й пример появляется  $w_i$  раз, а для учета нецелочисленных значений весов используется рандомизация.



*Рис. 18.6. Схема, показывающая, что обучение ансамбля позволяет добиться повышения выразительной мощи гипотез. Здесь представлены три линейные пороговые гипотезы, каждая из которых формирует положительную классификацию на незаштрихованной стороне, а в целом как положительные классифицируются все примеры, которые являются положительными согласно всем трем гипотезам. Полученная в итоге треугольная область представляет собой гипотезу, которая не может быть выражена в первоначальном пространстве гипотез*

Процедура усиления начинается с задания  $w_j=1$  для всех примеров (т.е. с обычного обучающего множества). На основании этого множества вырабатывается первая гипотеза  $h_1$ , которая классифицирует одни обучающие примеры правильно, а другие — неправильно. Желательно, чтобы следующая гипотеза лучше справлялась с неправильно классифицированными примерами, поэтому веса последних увеличиваются, а веса правильно классифицированных примеров уменьшаются. По этому обучающему множеству со вновь назначенными весами вырабатывается гипотеза  $h_2$ . Описанный процесс продолжается таким же образом до тех пор, пока не будет выработано  $M$  гипотез, где  $M$  становится входом для алгоритма усиления. Окончательная гипотеза-ансамбль представляет собой взвешенную мажоритарную комбинацию из всех  $M$  гипотез, каждой из которых назначен вес, соответствующий тому, насколько высокую производительность она показала при обработке обучающего множества. На рис. 18.7 показана концептуальная иллюстрация работы алгоритма. Эта основная идея усиления имеет много вариантов, в которых применяются различные способы корректировки весов и комбинирования гипотез. В листинге 18.2 показан один из конкретных алгоритмов, называемый AdaBoost. Хотя подробные сведения о том, как происходит корректировка весов, не имеет столь важного значения, алгоритм AdaBoost обладает очень важным свойством: если входной обучающий алгоритм  $L$  является **слабым обучающим** алгоритмом (это означает, что  $L$  всегда возвращает гипотезу со взвешенной ошибкой на обучающем множестве, которая лишь ненамного лучше по сравнению со случайным угадыванием, например, равным 50% при булевой классификации), то алгоритм AdaBoost при достаточно большом значении  $M$  возвращает гипотезу, идеально классифицирующую обучающие данные. Таким

образом, этот алгоритм значительно повышает точность первоначального обучающего алгоритма применительно к обучающим данным. Такой результат остается в силе независимо от того, насколько невыразительным является первоначальное пространство гипотез, а также от того, насколько сложна изучаемая функция.



*Рис. 18.7. Иллюстрация работы алгоритма усиления. Каждый затененный прямоугольник соответствует некоторому примеру; высота прямоугольника соответствует весу. Галочки и крестики показывают, был ли данный пример классифицирован правильно с помощью текущей гипотезы. Размер дерева решений показывает вес этой гипотезы в окончательном ансамбле*

**Листинг 18.2.** Один из вариантов метода усиления для обучения ансамбля, представленный в виде алгоритма **AdaBoost**. Этот алгоритм вырабатывает гипотезу, последовательно корректируя веса обучающих примеров. Функция **Weighted-Majority** вырабатывает гипотезу, которая возвращает выходное значение с наивысшими результатами голосования из числа гипотез, относящихся к вектору  $h$ , где результаты голосования взвешиваются с помощью вектора  $z$

```

function AdaBoost(examples, L, M) returns взвешенная мажоритарная
    комбинация гипотез
    inputs: examples, множество из N размеченных
        примеров  $(x_1, y_1), \dots, (x_N, y_N)$ 
        L, обучающий алгоритм
        M, количество гипотез в ансамбле
    local variables: w, вектор из N весов примеров, первоначально
        равных  $1/N$ 
        h, вектор из M гипотез
        z, вектор из M весов гипотез
    for m = 1 to M do
        h[m]  $\leftarrow$  L(examples, w)
        error  $\leftarrow$  0
        for j = 1 to N do
    
```

```

if  $h[m](x_j) \neq y_j$  then  $error \leftarrow error + w[j]$ 
for  $j = 1$  to  $N$  do
    if  $h[m](x_j) = y_j$  then  $w[j] \leftarrow w[j] \cdot error / (1 - error)$ 
     $w \leftarrow Normalize(w)$ 
 $z[m] \leftarrow \log(1 - error) / error$ 
return Weighted-Majority( $h$ ,  $z$ )

```

Рассмотрим, насколько хорошо метод усиления действует применительно к данным о ресторане. Выберем в качестве первоначального пространства гипотез класс **одноузловых деревьев решений**, представляющих собой деревья решений только с одной проверкой, в корневом узле. Нижняя кривая, приведенная на рис. 18.8, *a*, показывает, что неусиленные одноузловые деревья решений не очень эффективно действуют применительно к этому набору данных, достигая производительности предсказания, составляющей только 81% в расчете на 100 обучающих примеров. А после применения метода усиления (при  $M=5$ ) производительность становится выше и достигает 93% после обработки 100 примеров.

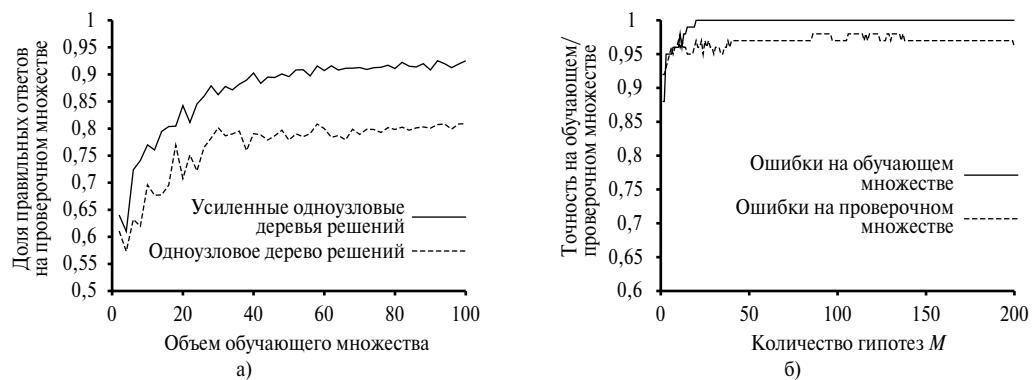


Рис. 18.8. Анализ производительности алгоритмов: график, показывающий, как изменяется производительность усиленных одноузловых деревьев решений при  $M=5$  по сравнению с неусиленными одноузловыми деревьями решений на примере данных о ресторане (*а*); доля правильных ответов, полученных на обучающем множестве и проверочном множестве, как функция от  $M$  (от количества гипотез в ансамбле) (*б*). Обратите внимание на то, что точность распознавания примеров из проверочного множества немного повышается даже после того, как точность распознавания примеров из обучающего множества достигает 1, т.е. после того, как ансамбль гипотез полностью согласуется с данными

По мере увеличения размера ансамбля  $M$  обнаруживается интересное явление. На рис. 18.8, *б* показана производительность обучающего множества (на 100 примерах) как функция от  $M$ . Обратите внимание на то, что ошибка достигает нуля (как и следует из определения метода усиления), когда  $M$  становится равным 20; это означает, что взвешенная мажоритарная комбинация из 20 одноузловых деревьев решений вполне позволяет определить точное соответствие для 100 примеров. По мере введения в ансамбль дополнительных одноузловых деревьев решений ошибка остается равной нулю. Этот график также показывает, что производительность обработки проверочного множества продолжает возрастать в течение долгого времени после того, как ошибка на обучающем множестве достигает нуля. При  $M=20$  производи-

тельность на проверочном множестве равна 0,95 (что соответствует 0,05 ошибки) и после чего увеличивается до 0,98 при таком большом значении, как  $M=137$ , прежде чем постепенно уменьшиться до 0,95.

Эта особенность, которая неизменно проявляется в самых разных наборах данных и пространствах гипотез, после ее обнаружения впервые показалась исследователям весьма неожиданной. Согласно принципу бритвы Оккама, не следует создавать гипотезы, более сложные, чем необходимо, а этот график говорит нам о том, что по мере усложнения гипотезы-ансамбля предсказания улучшаются! Для объяснения этого феномена было предложено несколько трактовок. Один из подходов к анализу такого явления состоит в том, что в процессе усиления аппроксимируется **байесовское обучение** (см. главу 20), притом что можно доказать, что байесовский алгоритм является оптимальным обучающим алгоритмом, а аппроксимация улучшается по мере введения дополнительных гипотез. Еще одно возможное объяснение состоит в том, что введение дополнительных гипотез позволяет добиться того, что ансамбль проводит все более определенное различие между положительными и отрицательными примерами, а это свойство способствует лучшей классификации новых примеров.

## 18.5. ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ АЛГОРИТМОВ ОБУЧЕНИЯ: ТЕОРИЯ ВЫЧИСЛИТЕЛЬНОГО ОБУЧЕНИЯ

Один из важных вопросов, поставленных в разделе 18.2, на который не был получен ответ, состоял в следующем: как можно убедиться в том, что в результате применения разработанного кем-то обучающего алгоритма была создана теория, позволяющая правильно предсказывать будущее? Формально этот вопрос можно переформулировать следующим образом: как определить, насколько гипотеза  $h$  близка к целевой функции  $f$ , если неизвестно, какой явлется сама функция  $f$ ? Подобные вопросы были предметом размышлений ученых в течение нескольких столетий. До тех пор, пока на них не будут получены ответы, машинное обучение в лучшем случае может рассматриваться лишь как научная область, причины успешных достижений которой остаются необъяснимыми.

Подход, принятый в данном разделе, основан на **теории вычислительного обучения** — научной области, которая находится на стыке искусственного интеллекта, статистики и теоретических компьютерных наук. Принцип, лежащий в ее основе, состоит в следующем: ~~любая~~ любая гипотеза, которая содержит серьезные ошибки, почти наверняка будет “открыта” с большой вероятностью после обработки небольшого количества примеров, поскольку она дает неправильные предсказания. Поэтому любая гипотеза, согласованная с достаточно большим множеством обучающих примеров, с меньшей вероятностью будет содержать серьезные ошибки; это означает, что она обязательно будет ~~приблизительно правильной с определенной вероятностью~~ **приблизительно правильной с определенной вероятностью**. Любой обучающий алгоритм, вырабатывающий гипотезы, которые с определенной вероятностью являются приблизительно правильными (Probably Approximately Correct — PAC), называется алгоритмом **PAC-обучения**.

При анализе приведенных выше доводов необходимо учитывать некоторые нюансы. Основной вопрос состоит в том, какова связь между обучающими и проверочными примерами; в конечном итоге желательно, чтобы гипотеза была приблизительно правильной

применительно к проверочному множеству, а не только к обучающему множеству. Основное предположение состоит в том, что и обучающее, и проверочное множества примеров извлекаются случайно и независимо друг от друга из одной и той же популяции примеров с одним и тем же распределением вероятностей. Это предположение называется предположением о **стационарности**. Если не принято предположение о стационарности, то теория вычислительного обучения не позволяет формулировать вообще какие-либо утверждения о будущем, поскольку не определена необходимая связь между будущим и прошлым. Предположение о стационарности равносильно тому предположению, что процесс, в котором осуществляется отбор примеров, не подвержен неблагоприятному влиянию. Очевидно, что если обучающее множество состоит только из надуманных примеров (например, фотографий двухголовых собак), то обучающий алгоритм не сможет сделать ничего иного, кроме как предложить безуспешные обобщения, касающиеся того, как распознавать обычных собак.

### Оценка количества необходимых примеров

Для того чтобы перевести эти предположения на практическую почву, необходимо ввести некоторые описанные ниже обозначения.

- Обозначим символом **X** множество всех возможных примеров.
- Обозначим символом **D** распределение, из которого извлекаются примеры.
- Обозначим символом **H** множество возможных гипотез.
- Обозначим символом **N** количество примеров в обучающем множестве.

Первоначально будем предполагать, что истинная функция  $f$  является элементом множества **H**. Теперь можно определить **ошибку** гипотезы  $h$  применительно к истинной функции  $f$ , если дано распределение вероятностей  $D$  по примерам, описывающее вероятность того, что гипотеза  $h$  отлична от функции  $f$  на некотором примере:

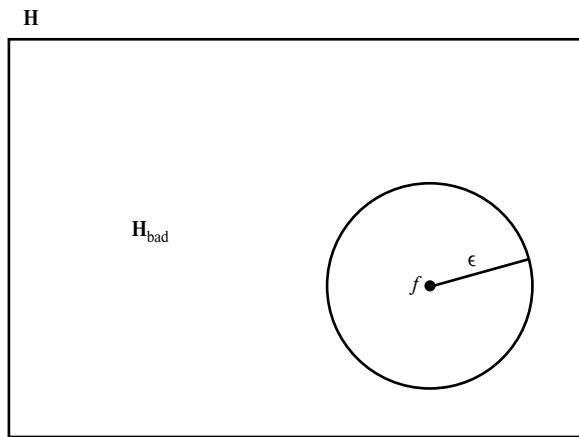
$$\text{error}(h) = P(h(x) \neq f(x) | x \text{ извлечен из } D)$$

Это — такое же количество, которое измерялось экспериментально с помощью кривых обучения, описанных выше в данной главе.

Гипотеза  $h$  называется **приблизительно правильной**, если  $\text{error}(h) \leq \epsilon$ , где  $\epsilon$  — небольшая константа. Примем к действию план решения этой проблемы, который состоит в том, чтобы доказать, что после просмотра  $N$  примеров все совместимые гипотезы с высокой вероятностью станут приблизительно правильными. Приблизительно правильная гипотеза может рассматриваться как “близкая” к истинной функции в пространстве гипотез: она находится внутри так называемого  **$\epsilon$ -шара**, который окружает истинную функцию  $f$ . На рис. 18.9 показано множество всех гипотез **H**, которое подразделяется на  $\epsilon$ -шар, окружающий функцию  $f$ , и все остальные гипотезы, принадлежащие к множеству, которое мы будем называть  $H_{\text{bad}}$ .

Вероятность того, что гипотеза  $h_b \in H_{\text{bad}}$ , содержащая “серезную ошибку”, будет согласована с первыми  $N$  примерами, можно вычислить следующим образом. Известно, что  $\text{error}(h_b) > \epsilon$ . В таком случае вероятность того, что эта гипотеза согласуется с заданным примером, равна по меньшей мере  $1 - \epsilon$ . Границное значение этой вероятности для  $N$  примеров равно:

$$P(h_b \text{ согласуется с } N \text{ примерами}) \leq (1 - \epsilon)^N$$



*Рис. 18.9. Схематическое изображение пространства гипотез, на котором показан  $\varepsilon$ -шар, окружающий истинную функцию  $f$*

Вероятность того, что множество  $H_{\text{bad}}$  содержит по меньшей мере одну совместимую гипотезу, ограничивается суммой отдельных вероятностей:

$$P(H_{\text{bad}} \text{ содержит совместимую гипотезу}) \leq |H_{\text{bad}}| (1-\varepsilon)^N \leq |\mathbf{H}| (1-\varepsilon)^N$$

где учитывался тот факт, что  $|H_{\text{bad}}| \leq |\mathbf{H}|$ . Желательно уменьшить вероятность этого события так, чтобы она не превышала некоторого небольшого числа  $\delta$ :

$$|\mathbf{H}| (1-\varepsilon)^N \leq \delta$$

С учетом того, что  $1-\varepsilon \leq e^{-\varepsilon}$ , такой цели можно добиться, предоставив алгоритму возможность обработать следующее количество примеров:

$$N \geq \frac{1}{\varepsilon} (\ln \frac{1}{\delta} + \ln |\mathbf{H}|) \quad (18.1)$$

Таким образом, если обучающий алгоритм возвратит гипотезу, совместимую с этим или большим количеством примеров, то с вероятностью, по меньшей мере равной  $1-\delta$ , он будет иметь, самое большое, ошибку  $\varepsilon$ . Иными словами, полученная гипотеза будет по всей вероятности приблизительно правильной. Количество требуемых примеров, зависящее от  $\varepsilon$  и  $\delta$ , называется **выборочной сложностью** (sample complexity) пространства гипотез.

Поэтому создается впечатление, что основной вопрос сводится к определению размера пространства гипотез. Как было показано выше, если  $\mathbf{H}$  — множество всех булевых функций от  $n$  атрибутов, то  $|\mathbf{H}| = 2^{2^n}$ . Таким образом, выборочная сложность пространства растет в зависимости от  $2^n$ . Поскольку количество возможных примеров также равно  $2^n$ , на этом основании можно утверждать, что любой обучающий алгоритм для пространства всех булевых функций не может функционировать лучше, чем поисковая таблица, если он просто возвращает гипотезу, совместимую со всеми известными примерами. Еще один способ убедиться в справедливости этого утверждения может быть основан на том наблюдении, что для любого не встречавшегося ранее примера пространство гипотез будет содержать столько же со-

вместимых гипотез, которые предсказывают положительный результат, сколько имеется гипотез, предсказывающих отрицательный результат.

Поэтому дилемма, с которой мы сталкиваемся, состоит в следующем: если пространство функций, которые могут рассматриваться алгоритмом, останется неограниченным, то алгоритм не будет способен к обучению, но, с другой стороны, ограничение пространства может привести к полному удалению истинной функции. Существуют два способа “разрешения” этой дилеммы. Первый способ состоит в соблюдении требования, чтобы алгоритм возвращал не просто любую совместимую гипотезу, а преимущественно наиболее простую гипотезу (что и предусмотрено в обучении деревьев решений). Теоретический анализ подобных алгоритмов выходит за рамки данной книги, но в тех случаях, когда задача поиска простых совместимых гипотез является легко разрешимой, результаты определения выборочной сложности обычно бывают лучше по сравнению с результатами анализа, основанными только на определении совместимости. Второй способ разрешения дилеммы, который рассматривается ниже, состоит в том, что нужно сосредоточиться на тех подмножествах всего множества булевых функций, которые доступны для обучения. Идея этого способа состоит в том, что в большинстве случаев не требуется полная выразительная мощь булевых функций и можно ограничиться использованием более простых языков. В следующем разделе один из таких ограниченных языков рассматривается более подробно.

## Обучение списков решений

Список решений — это логическое выражение с ограниченной формой. Он состоит из ряда проверок, каждая из которых представляет собой конъюнкцию литералов. Если проверка, применяемая к описанию примера, завершается успешно, то список решений задает возвращаемое значение. Если же проверка оканчивается неудачей, то обработка продолжается со следующей проверки в списке<sup>6</sup>. Списки решений напоминают деревья решений, но их общая структура проще. С другой стороны, отдельные проверки в них намного сложнее. На рис. 18.10 показан список решений, который представляет следующую гипотезу:

$$\forall x \text{WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some}) \vee (\text{Patrons}(x, \text{Full}) \wedge \text{Fri/Sat}(x))$$

Если допускается применение проверок с произвольными размерами, то списки решений становятся способными представить любую булеву функцию (упр. 18.15). С другой стороны, если размер каждой проверки ограничен, самое большое,  $k$  литералами, то для обучающего алгоритма появляется возможность успешно создавать обобщения на основе небольшого количества примеров. Мы будем называть такой язык списков решений с  $k$  литералами языком  $\bowtie k\text{-DL}$  (DL — Decision List). Пример, приведенный на рис. 18.10, относится к языку 2-DL. Можно легко показать (упр. 18.15), что язык  $k$ -DL включает в качестве подмножества язык  $\bowtie k\text{-DT}$  (DT — Decision Tree), который представляет собой множество всех деревьев решений с глубиной, не превышающей  $k$ . Важно помнить, что конкретный язык, к которому применяется обозначение  $k$ -DL, зависит от атрибутов, используемых для описания примеров. Мы будем использовать запись  $k\text{-DL}(n)$  для обозначения языка  $k$ -DL, в котором применяется  $n$  булевых атрибутов.

<sup>6</sup> Поэтому список решений идентичен по своей структуре оператору COND в языке Lisp.

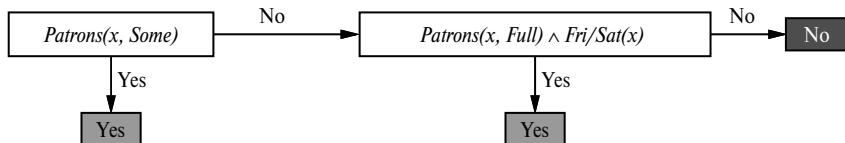


Рис. 18.10. Список решений для задачи с рестораном

Первая задача состоит в том, чтобы показать, что язык k-DL доступен для изучения, т.е. что любая функция, заданная в языке k-DL, может быть точно аппроксимирована после обучения на приемлемом количестве примеров. Для этого необходимо рассчитать количество гипотез, которые могут быть представлены на этом языке. Допустим, что язык проверок (конъюнкций из самое большое  $k$  литералов, в которых используется  $n$  атрибутов) обозначается как  $\text{Conj}(n, k)$ . Поскольку любой список решений состоит из проверок, а каждая проверка может быть связана с результатом *Yes* или *No* или может отсутствовать в списке решений, то количество различных множеств проверок, из которых могут состоять гипотезы, измеряется величиной  $3^{|\text{Conj}(n, k)|}$ . Каждое из этих множеств проверок может быть задано в любом порядке, поэтому справедливо следующее соотношение:

$$|k\text{-DL}(n)| \leq 3^{|\text{Conj}(n, k)|} |\text{Conj}(n, k)|!$$

Количество конъюнкций из  $k$  литералов с  $n$  атрибутами может быть определено таким образом:

$$|\text{Conj}(n, k)| = \sum_{i=0}^k \binom{2n}{i} = O(n^k)$$

Поэтому после проведения некоторых преобразований можно получить следующее:

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}$$

Это соотношение можно подставить в уравнение 18.1, чтобы показать, что количество примеров, необходимое для PAC-обучения некоторой функции k-DL, определяется полиномиальной зависимостью от  $n$ :

$$N \geq \frac{1}{\epsilon} \ln \frac{1}{\delta} + O(n^k \log_2(n^k))$$

Поэтому любой алгоритм, возвращающий согласованный список решений, будет обеспечивать PAC-обучение некоторой функции k-DL с помощью приемлемого количества примеров, если значение  $k$  невелико.

Следующая задача состоит в том, чтобы найти эффективный алгоритм, который возвращает совместимый список решений. Мы будем использовать жадный алгоритм, называемый *Decision-List-Learning*, который повторно отыскивает проверку, точно согласующуюся с некоторым подмножеством обучающего множества. Найдя такую проверку, алгоритм добавляет ее к создаваемому списку решений и удаляет соответствующие примеры. Затем алгоритм формирует только оставшуюся часть списка решений, используя лишь оставшиеся примеры. Такая операция повторяется до тех пор, пока не останется ни одного примера. Этот алгоритм показан в листинге 18.3.

### Листинг 18.3. Алгоритм обучения списков решений

```

function Decision-List-Learning(examples) returns список решений
    или индикатор неудачи failure

    if множество examples пусто
        then return тривиальный список решений No
    t  $\leftarrow$  проверка, которая согласуется с непустым подмножеством
        examplest множества examples, таким что все элементы examplest
        являются либо положительными, либо отрицательными примерами
    if такая проверка t отсутствует then return failure
    if примеры в подмножестве examplest являются положительными
        then o  $\leftarrow$  Yes
    else o  $\leftarrow$  No
    return список решений с начальной проверкой t, результатом о
        и оставшимися проверками, определяемыми выражением
        Decision-List-Learning(examples-examplest)

```

В этом алгоритме не определен метод выбора следующей проверки для добавления к списку решений. Хотя формально приведенные выше результаты не зависят от такого метода выбора, представляется резонным подход, позволяющий отдавать предпочтение небольшим проверкам, которые согласуются с большими множествами однородно классифицируемых примеров, для того чтобы общий список решений был как можно более компактным. Простейшая стратегия достижения этой цели состоит в том, что нужно отыскивать наименьшую проверку *t*, которая согласуется с любым однородно классифицируемым подмножеством, независимо от размера этого подмножества. Как показано на рис. 18.11, даже такой примитивный подход действует вполне успешно.



Рис. 18.11. Кривая обучения по данным о ресторане при использовании алгоритма Decision-List-Learning. Для сравнения показана кривая, соответствующая алгоритму Decision-Tree-Learning

### Обсуждение полученных результатов

В рамках теории вычислительного обучения был создан новый способ трактовки проблемы обучения. В начале 1960-х годов разработки в области теории обучения

сосредоточивались в основном на проблеме успешной **идентификации в пределе**. В соответствии с этим понятием любой алгоритм идентификации должен возвращать гипотезу, которая точно совпадает с истинной функцией. Один из способов решения этой задачи состоит в следующем: вначале упорядочить все гипотезы в множестве **Н** в соответствии с некоторым критерием простоты, затем выбрать простейшую гипотезу, совместимую со всеми полученными до сих пор примерами. По мере получения новых примеров в этом методе предусмотрен отказ от более простой гипотезы, которая стала недействительной, и принятие вместо нее более сложной гипотезы. Но после того как достигается истинная функция, от нее никогда не происходит отказ. К сожалению, в больших пространствах гипотез количество примеров и продолжительность вычислений, необходимая для достижения истинной функции, становятся колоссальными. Поэтому теория вычислительного обучения не определяет безусловного требования, чтобы обучающийся агент определил “единственный истинный закон”, руководящий его средой, а вместо этого допускает, чтобы он нашел гипотезу с определенной степенью прогностической точности. Кроме того, в теории вычислительного обучения большое внимание уделяется выявлению компромисса между выразительностью языка гипотез и сложностью обучения, поэтому развитие этой теории непосредственно привело к созданию важного класса обучающих алгоритмов, получившего название *машин поддерживающих векторов* (support vector machine).

Приведенные в данном разделе результаты РАС-обучения представляют собой результаты определения сложности для наихудшего случая и не обязательно отражают выборочную сложность для среднего случая, которая измеряется с помощью приведенных здесь кривых обучения. При анализе среднего случая необходимо также принять предположения о распределении вероятностей примеров и распределении вероятностей истинных функций, которые должны быть изучены с помощью данного алгоритма. По мере того как достигается все лучшее понимание этих проблем, теория вычислительного обучения предоставляет все более ценную помощь исследователям в области машинного обучения, для которых важно предсказать или модифицировать обучающую способность создаваемых ими алгоритмов. Кроме списков решений, важные результаты были получены почти для всех известных подклассов булевых функций, для множеств высказываний первого порядка (глава 19) и для нейронных сетей (глава 20). Эти результаты показывают, что чистые задачи индуктивного обучения, в которых агент приступает к работе без априорных знаний о целевой функции, обычно являются очень трудными. Как будет показано в главе 19, использование априорных знаний для управления индуктивным обучением дает возможность изучать весьма крупные множества высказываний на основе приемлемого количества примеров, даже если используется столь выразительный язык, как логика первого порядка.

## 18.6. РЕЗЮМЕ

Настоящая глава главным образом посвящена изложению темы индуктивного обучения детерминированных функций на основе примеров. Основные вопросы, рассматриваемые в этой главе, перечислены ниже.

- Обучение принимает много разных форм в зависимости от характера производительного элемента, компонента, подлежащего усовершенствованию, и доступной обратной связи.
- Если доступна обратная связь либо от учителя, либо от среды, позволяющая получать правильные значения, относящиеся к примерам, то задача обучения относится к типу **контролируемого обучения**. Такая задача, называемая также **индуктивным обучением**, сводится к изучению некоторой функции на примерах ее входных и выходных данных. Изучение функции с дискретными значениями называется **классификацией**; изучение непрерывной функции называется **регрессией**.
- Индуктивное обучение сводится к поиску **совместимой** гипотезы, которая согласуется с примерами. При этом должен соблюдаться принцип **бритвы Оккама**, согласно которому следует всегда выбирать наиболее простую совместимую гипотезу. Сложность решения этой задачи зависит от выбранного способа представления.
- **Деревья решений** позволяют представить любые булевые функции. Эвристика, определяющая **приращение информации**, может стать основой эффективного метода поиска простого, совместимого дерева решений.
- Производительность обучающего алгоритма измеряется **кривой обучения**, которая показывает прогностическую точность применения алгоритма к **проверочному множеству** как функцию от размера **обучающего множества**.
- Методы обучения ансамбля деревьев решений, такие как **усиление**, часто показывают более высокую производительность по сравнению с методами обучения отдельных деревьев решений.
- Для анализа выборочной и вычислительной сложности индуктивного обучения применяется **теория вычислительного обучения**. Эта теория позволяет найти компромисс между выразительностью языка гипотез и легкостью обучения.

---

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

---

История философских исследований в области индуктивного обучения описана в главе 1. Уильям из Окхема (1280–1349), наиболее влиятельный философ своей эпохи, внесший в средние века наибольший вклад в развитие эпистемологии, логики и метафизики, заслужил признание потомков за то, что выдвинул важный принцип, называемый “бритвой Оккама”, который на латыни выражается словами “*Entia non sunt multiplicanda praeter necessitatem*”, а в переводе звучит так: “Не следует множить сущности без необходимости”. К сожалению, это замечательное изречение не удалось найти в его трудах точно в такой формулировке.

Одной из первых систем, в которой использовались деревья решений (или, как они в ней именовались, **различительные сети** (*discrimination nets*)), была система EPAM (Elementary Perceiver And Memorizer) Фейгенбаума [457]. Система EPAM предназначалась для применения в качестве модели имитации познания в процессе обучения человека новым понятиям. В системе CLS [707] для формирования деревьев решений использовался эвристический прогностический метод. В системе ID3

[1256] была дополнительно реализована крайне важная идея о том, что для обеспечения функционирования эвристической функции может применяться информационное содержание. Сама теория информации была разработана Клодом Шенноном для использования в качестве средства исследования процессов связи [1394]. Важный вклад, сделанный Шенноном, состоит также в том, что он разработал первые образцы систем, действующих по принципу машинного обучения, в частности механическую мышь, получившую имя *Theseus* (Тезей), которая обучалась прохождению через лабиринт по методу проб и ошибок. Метод  $\chi^2$  отсечения ветвей деревьев был описан Куинланом [1257]. Описание пакета C4.5 производственного назначения, который основан на использовании деревьев решений, можно найти в [1259]. В литературе по статистике существует отдельное направление, посвященное исследованиям в области деревьев решений. Одним из основных источников информации по этой теме является книга *Classification and Regression Trees* [180], известная под названием книги “*CART*”.

Предпринимались также попытки применить многие другие алгоритмические подходы к обучению. Подход, основанный на использовании **текущей наилучшей гипотезы**, предусматривает сопровождение единственной гипотезы, а также ее уточнение, если обнаруживается, что она слишком широка, и обобщение, если она оказывается слишком узкой. Такой принцип проведения исследований уже давно применяется в философии [1049]. Кроме того, даже ранние работы в области когнитивной психологии показали, что в этом состоит естественная форма изучения концепций людьми [199]. В искусственном интеллекте такой подход наиболее тесно связан с работой Патрика Уинстона, в докторской диссертации которого [1602] рассматривается проблема изучения описаний сложных объектов. В методе **пространства версий** [1061], [1062] принят другой подход, в котором сопровождается множество всех совместимых гипотез и удаляются оказавшиеся несовместимыми с новыми примерами. Такой подход был реализован в экспертной системе Meta-Dendral, применяемой в химии [202], а затем — системе Митчелла Lex [1066], которая обучалась решению задач исчисления. Третье влиятельное направление возникло под влиянием работы Михальского и его коллег над рядом алгоритмов AQ, применявшимся для изучения множеств логических правил [1038], [1041].

Метод обучения ансамбля деревьев решений все чаще применяется для повышения производительности обучающих алгоритмов. Первый эффективный метод такого типа, называемый **созданием мульти множеств** [179], предусматривает формирование комбинаций гипотез, изученных на основе многочисленных наборов данных **начальной загрузки**, каждый из которых формируется путем выборки подмножества из первоначального множества данных. Метод **усиления**, описанный в данной главе, впервые был изложен в теоретической работе Шейпира [1361]. Алгоритм AdaBoost был разработан Фрейндом и Шейпиром [501], а результаты его теоретического анализа приведены в [1360]. В [505] принципы работы метода усиления описаны с точки зрения специалиста по статистике.

Теоретический анализ обучающих алгоритмов начался с работы Голда [569] по проблеме **идентификации в пределе**. Стимулом к развитию этого подхода отчасти явились модели научного открытия, разработанные в рамках философии науки [1229], но этот подход в основном применялся к задаче изучения грамматик на примерах предложений [1162].

Подход, основанный на изучении “идентификации в пределе”, в основном посвящен достижаемой в конечном итоге сходимости, а исследования **колмогоровской сложности**, или **алгоритмической сложности**, проведенные независимо Соломоновым [1445] и Колмогоровым [828], представляют собой попытку дать формальное определение понятия простоты, используемого в принципе бритвы Оккама. Чтобы исключить проблему, связанную с тем, что простота зависит от способа представления информации, было предложено измерять простоту как длину кратчайшей программы для универсальной машины Тьюринга, которая правильно воспроизводит наблюдаемые данные. Хотя существует много возможных универсальных машин Тьюринга и поэтому много возможных “кратчайших” программ, было обнаружено, что эти программы отличаются по длине не больше чем на некоторую константу, независимую от объема данных. Это — великолепное открытие, которое по сути показывает, что любое искажение в первоначальном представлении данных в конечном итоге должно быть преодолено с помощью самих же данных. Его широкому применению препятствует лишь то, что задача вычисления длины кратчайшей программы является неразрешимой. Однако вместо точного значения длины могут применяться такие приближенные критерии, как **минимальная длина описания**, или MDL (Minimum Description Length) [1291], которые позволяют добиться на практике превосходных результатов. Наилучшим источником сведений по проблеме колмогоровской сложности является книга Ли и Витаны [927].

Теория вычислительного обучения (т.е. теория PAC-обучения) была выдвинута Лесли Валиантом [1528]. В работе Валианта подчеркивается важность вычислительной и выборочной сложности. Совместно с Майклом Кернсом [783] Валиант показал, что задача PAC-обучения некоторых классов концепций является трудноразрешимой, даже если в примерах присутствует достаточный объем информации. Некоторые положительные результаты были получены для таких классов, как списки решений [1293].

В статистике сложилось независимое направление анализа выборочной сложности, начиная с одной важной работы Вапника и Червоненкиса по **теории равномерной сходимости** [1538]. Понятие так называемой **VC-размерности** (VC — сокращение от Vapnik–Chervonenkis) позволяет получить критерий, приблизительно аналогичный, но более общий, чем критерий  $\ln |\mathcal{H}|$ , полученный на основе анализа PAC-обучения. В частности, критерий VC-размерности может применяться к непрерывным классам функций, на которые не распространяется стандартный анализ PAC-обучения. Теория PAC-обучения и теория VC-размерности были впервые связаны в единое целое группой ученых, называемых “четырьмя немцами” (хотя ни один из них фактически не является немцем), — Блумером, Эренфойхтом, Хаусслером и Вармутом [141]. Дальнейшие исследования в области теории VC-размерности привели к открытию понятия **машины поддерживающих векторов**, или SVM (Support Vector Machine) [156], [1537], которая будет описана в главе 20.

Большое количество важных статей по машинному обучению было собрано в книге *Readings in Machine Learning* [1399]. Кроме того, много важных статей, а также огромные библиографические справочники содержатся в двух томах, *Machine Learning 1* [1039] и *Machine Learning 2* [1040]. В [1565] дано широкое введение в область методов изучения функций, применяемых в машинном обучении, статистике и нейронных сетях. Исследования, намного превосходящие по своей широте все прочие работы в области сравнения производительности обучающих алгоритмов,

были проведены в рамках проекта Statlog [1047]. Результаты продуктивных современных исследований по машинному обучению публикуются в ежегодном сборнике трудов конференций *International Conference on Machine Learning* и *Conference on Neural Information Processing Systems*, в журналах *Machine Learning* и *Journal of Machine Learning Research*, а также в основных журналах по искусственному интеллекту. Кроме того, труды в области теории вычислительного обучения публикуются в материалах ежегодного семинара *ACM Workshop on Computational Learning Theory* (COLT), а само это направление представлено в книгах [34] и [786].

## УПРАЖНЕНИЯ

- 18.1.** Рассмотрим проблему, стоящую перед младенцем, который учится говорить и понимать язык. Объясните, как этот процесс вписывается в общую модель обучения, указывая по мере необходимости каждый из компонентов этой модели.
- 18.2.** Повторите упр. 18.1 для случая обучения игре в теннис (или обучения какому-то другому виду спорта, с которым вы знакомы). Относится ли оно по своему типу к контролируемому обучению или обучению с подкреплением?
- 18.3.** Составьте дерево решений для задачи принятия решения о том, следует ли двигаться вперед на уличном перекрестке в той ситуации, когда на светофоре только что загорелся зеленый свет.
- 18.4.** Мы никогда не проверяем дважды один и тот же атрибут вдоль одного пути в дереве решений. Почему этого не делается?
- 18.5.** Предположим, что на основе дерева решений сформировано обучающее множество, а затем к этому обучающему множеству применен алгоритм обучения дерева решений. Можно ли рассчитывать на то, что этот обучающий алгоритм в конечном итоге возвратит правильное дерево, по мере того как размер обучающего множества будет стремиться к бесконечности? Почему да или почему нет?
- 18.6.** Хороший обучающий алгоритм со “спарринг-партнером” состоит в следующем. Вначале создать таблицу из всех обучающих примеров и выяснить, какие выходные результаты встречаются наиболее часто среди всех обучающих примеров; назовем их  $d$ . Затем, после получения входных данных, которые отсутствуют в таблице, возвращать просто данные из множества  $d$ , а для входных данных, которые имеются в таблице, возвращать связанные с ними выходные данные (если же количество выходных данных больше единицы, возвращать наиболее часто встречающиеся выходные данные). Реализуйте этот алгоритм и определите, насколько хорошо он действует в проблемной области задачи с рестораном. Это позволит вам получить представление о точке отсчета для этой проблемной области — о минимальной производительности, которую должен быть способным достичь любой алгоритм.
- 18.7.** Предположим, что вы проводите эксперименты по обучению с помощью нового алгоритма. У вас имеется набор данных, состоящий из 25 примеров каждого из двух классов. Вы планируете использовать перекрестную проверку с исключением одного примера и для определения точки отсчета выполняете прогон вашей экспериментальной системы на простом мажоритарном клас-

сификаторе. (Мажоритарный классификатор получает на входе множество обучающих данных, а затем всегда выводит тот класс, который имеет преимущественное распространение в обучающем множестве, независимо от входных данных.) Вы предполагаете, что мажоритарный классификатор в среднем должен позволить получить оценку около 50% при перекрестной проверке с исключением одного примера, но, к вашему удивлению, он выдает оценку нуль. Можете ли вы объяснить причины, по которым получен такой результат?

- 18.8.** При рекурсивном формировании деревьев решений иногда происходит так, что в листовом узле остается смешанное множество положительных и отрицательных примеров, даже несмотря на то, что были использованы все атрибуты. Предположим, что количество положительных примеров равно  $p$ , а количество отрицательных примеров —  $n$ .
- Покажите, что решение, используемое в алгоритме Decision-Tree-Learning, в котором выбирается мажоритарная классификация, минимизирует абсолютную ошибку на множестве примеров в листовом узле.
  - Покажите, что ~~вероятность класса  $p/(p+n)$~~  минимизирует сумму квадратичных ошибок.
- 18.9.** Предположим, что в обучающем алгоритме предпринимается попытка найти совместимую гипотезу, притом что фактически классификации примеров являются случайными. Имеется  $n$  булевых атрибутов, а примеры извлекаются с помощью равномерного распределения из множества  $2^n$  возможных примеров. Рассчитайте количество примеров, которые требуется получить, прежде чем вероятность обнаружения противоречия в данных достигнет 0,5.
- 18.10.** Предположим, что некоторый атрибут разбивает множество примеров  $E$  на подмножества  $E_i$  и что каждое подмножество включает  $p_i$  положительных примеров и  $n_i$  отрицательных примеров. Покажите, что этот атрибут строго обеспечивает положительное приращение информации, если отношение  $p_i / (p_i + n_i)$  не является одинаковым для всех  $i$ .
- 18.11.** Модифицируйте алгоритм Decision-Tree-Learning, для того чтобы в нем применялось  $\chi^2$ -отсечение. Для ознакомления с подробными сведениями по этой теме вам может потребоваться обратиться к [1257].
- 18.12.** В стандартном алгоритме Decision-Tree-Learning, описанном в данной главе, не учитываются такие случаи, в которых отдельные примеры имеют отсутствующие значения атрибутов.
- Прежде всего необходимо найти способ классификации таких примеров, если дано дерево решений, включающее проверки по атрибутам, для которых могут отсутствовать значения. Предположим, что в некотором примере  $X$  отсутствует значение для атрибута  $A$  и что в дереве решений выполняется проверка атрибута  $A$  в узле, достигнутом примером  $x$ . Один из способов справиться с такой ситуацией состоит в том, чтобы предложить, что в этом примере заданы все возможные значения для данного атрибута, но взвесить каждое значение с учетом его частоты среди всех примеров, достигших этого узла в дереве решений. Алгоритм классификации должен проследовать по всем ветвям в каждом узле, для которого

отсутствует некоторое значение, и умножить полученные значения на веса вдоль каждого пути. Напишите модифицированный алгоритм классификации для деревьев решений, который действует по такому принципу.

- 6) Теперь необходимо внести изменения в расчеты приращения информации таким образом, чтобы в любой конкретной коллекции примеров  $C$  в любом конкретном узле дерева в процессе его формирования в качестве значений любого из оставшихся атрибутов присваивались “предполагаемые” величины, соответствующие частотам этих значений в множестве  $C$ .
- 18.13.** В данной главе было отмечено, что при использовании атрибутов со многими различными возможными значениями могут возникать проблемы, связанные с применением критерия приращения информации. Такие атрибуты, как правило, разбивают примеры на многочисленные небольшие классы или даже одноэлементные классы, из-за чего они начинают казаться в высшей степени релевантными в соответствии с критерием приращения информации. **Критерий коэффициента приращения** позволяет выбирать атрибуты с учетом отношения между соответствующим им приростом и свойственным им информационным содержанием, т.е. количеством информации, содержащемся в ответе на вопрос: “Каково значение этого атрибута?” Поэтому при использовании критерия коэффициента приращения предпринимается попытка измерить, насколько полно этот атрибут предоставляет информацию о правильной классификации некоторого примера. Запишите математическое выражение для информационного содержания атрибута и реализуйте критерий коэффициента приращения в алгоритме Decision-Tree-Learning.
- 18.14.** Рассмотрите алгоритм обучения ансамбля деревьев решений, в котором используется простое мажоритарное голосование среди  $M$  изучаемых гипотез. Предположим, что каждая гипотеза имеет ошибку  $\epsilon$  и что ошибки, допущенные в каждой гипотезе, не зависят от ошибок в других гипотезах. Составьте формулу для ошибки в алгоритме обучения ансамбля в терминах  $M$  и  $\epsilon$  и проведите по ней вычисления для тех случаев, когда  $M=5, 10$  и  $20$ , а  $\epsilon=0.1, 0.2$  и  $0.4$ . Возможно ли, что ошибка алгоритма обучения ансамбля будет хуже, чем  $\epsilon$ , если предположение о независимости будет исключено?
- 18.15.** Это упражнение касается вопроса о выразительности списков решений (раздел 18.5).
- Покажите, что списки решений позволяют представить любую булеву функцию, если размер проверок не ограничен.
  - Покажите, что если проверки могут включать самое большое  $k$  литералов каждая, то списки решений позволяют представить любую функцию, которая может быть представлена с помощью дерева решений с глубиной  $k$ .

# 19 ПРИМЕНЕНИЕ ЗНАНИЙ В ОБУЧЕНИИ

*В этой главе рассматривается задача обучения в тех условиях, когда уже кое-что известно.*

Во всех подходах к обучению, описанных в предыдущих трех главах, основная идея состоит в том, что должна быть сформирована функция, входные и выходные данные которой соответствуют закономерностям, наблюдаемым в данных. В каждом подобном случае методы обучения можно рассматривать как поиск в пространстве гипотез для обнаружения подходящей функции, начиная только с очень простых предположений о форме этой функции, таких как “полином второго порядка” или “дерево решений”, и руководствуясь такими критериями, как “чем проще, тем лучше”. Применение такого подхода равносильно требованию, что перед началом изучения чего-либо нового необходимо в первую очередь забыть (почти) все, что вы знаете. А в данной главе рассматриваются методы обучения, позволяющие воспользоваться  **априорными знаниями** о мире. В большинстве случаев априорные знания представлены в виде общих логических теорий первого порядка, поэтому вначале в этой главе необходимо систематизировать сведения о представлении знаний и обучении.

## 19.1. ЛОГИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ ОБУЧЕНИЯ

В главе 18 было дано определение чисто индуктивного обучения как процесса поиска гипотезы, которая согласуется с наблюдаемыми примерами. В данной главе это определение уточняется и распространяется на тот случай, когда гипотеза представлена в виде множества логических высказываний. Описания примеров и определения классов также заданы в виде логических высказываний, а классификация нового примера может быть выполнена путем логического вывода классификационного высказывания из гипотезы и описания примера. Такой подход обеспечивает инкрементное формирование гипотез путем последовательного добавления каждый раз по одному предложению. Он также дает возможность использовать априорные знания, поскольку уже известные высказывания могут помочь при классификации новых примеров. На первый взгляд может показаться, что логическая формулировка

задачи обучения требует выполнения вначале большого объема дополнительной работы, но, как оказалось, она позволяет прояснить многие нерешенные проблемы обучения. Указанный подход дает возможность намного превзойти простые методы обучения, описанные в главе 18, поставив на службу обучению всю мощь логического вывода.

## Примеры и гипотезы

Вернемся к описанной в главе 18 задаче обучения с рестораном, в которой нужно было изучить правило принятия решения о том, при каких условиях следует ждать освобождения столика. В этой главе для описания примеров применялись **атрибуты**, такие как *Alternate*, *Bar*, *Fri/Sat* и т.д. В логической формулировке задачи любой пример представляет собой объект, для описания которого используется логическое высказывание, а атрибуты становятся унарными предикатами. Примем общее обозначение  $X_i$  для  $i$ -го примера. В частности, первый пример, приведенный в табл. 18.1, может быть описан с помощью таких высказываний:

$$\text{Alternate}(X_1) \wedge \neg\text{Bar}(X_1) \wedge \neg\text{Fri/Sat}(X_1) \wedge \text{Hungry}(X_1) \wedge \dots$$

Обозначение  $D_i(X_i)$  будет использоваться для ссылки на описание  $X_i$ , где  $D_i$  может представлять собой любое логическое выражение, принимающее один параметр. Классификация объекта определяется примерно таким высказыванием:

$$\text{WillWait}(X_1)$$

Общее обозначение  $Q(X_i)$  будет применяться, если пример является положительным, а  $\neg Q(X_i)$  — если отрицательным. В таком случае полное обучающее множество представляет собой конъюнкцию всех описательных и классификационных высказываний.

Целью индуктивного обучения в логической постановке задачи является поиск эквивалентного логического выражения для целевого предиката  $Q$ , который может использоваться для правильной классификации примеров. Подобное выражение, которое мы будем называть **потенциальным определением** целевого предиката, предлагается в каждой гипотезе. Используя  $C_i$  для обозначения потенциального определения, можно утверждать, что каждая гипотеза  $H_i$  представляет собой высказывание в форме  $\forall x Q(x) \Leftrightarrow C_i(x)$ . В частности, дерево решений представляет собой утверждение, что целевой предикат принимает истинное значение по отношению к какому-то объекту тогда и только тогда, когда выполняются условия в одной из ветвей, ведущих к листовому узлу со значением *true*. Таким образом, на рис. 18.4 в графической форме выражено следующее логическое определение (которому мы присвоим обозначение  $H_r$  для использования его в будущем):

$$\begin{aligned} \forall r \text{ WillWait}(r) \Leftrightarrow & \text{Patrons}(r, \text{Some}) \\ \vee & \text{Patrons}(r, \text{Full}) \wedge \text{Hungry}(r) \wedge \text{Type}(r, \text{French}) \\ \vee & \text{Patrons}(r, \text{Full}) \wedge \text{Hungry}(r) \wedge \text{Type}(r, \text{Thai}) \\ & \wedge \text{Fri/Sat}(r) \\ \vee & \text{Patrons}(r, \text{Full}) \wedge \text{Hungry}(r) \wedge \text{Type}(r, \text{Burger}) \end{aligned} \quad (19.1)$$

Каждая гипотеза предсказывает, что некоторое множество примеров (а именно множество тех примеров, которые соответствуют ее потенциальному определению) будет представлять собой примеры целевого предиката. Такое множество называется

☞ **расширением** предиката. Это означает, что две гипотезы с разными расширениями являются логически несовместимыми друг с другом, поскольку они не согласуются в своих предсказаниях по меньшей мере в одном примере. А если две гипотезы имеют одно и то же расширение, они логически эквивалентны.

Пространство гипотез **H** алгоритма обучения представляет собой множество всех гипотез  $\{H_1, \dots, H_n\}$ , для поддержки которых предназначен данный конкретный обучающий алгоритм. Например, алгоритм Decision-Tree-Learning способен поддерживать любую гипотезу в дереве решений, определенную в терминах заранее предусмотренных атрибутов, поэтому пространство гипотез этого алгоритма состоит из всех таких деревьев решений. Предполагается, что алгоритм обучения позволяет с полной уверенностью утверждать, что одна из гипотез является правильной; это означает, что данный алгоритм выражает определенную степень уверенности в истинности следующего высказывания:

$$H_1 \vee H_2 \vee H_3 \vee \dots \vee H_n \quad (19.2)$$

По мере поступления новых примеров появляется возможность исключать гипотезы, не **совместимые** с этими примерами. Рассмотрим более внимательно затронутое здесь понятие совместимости. Очевидно, что если гипотеза  $H_i$  совместима со всем обучающим множеством, то она должна быть совместимой с каждым примером. А что повлекла бы за собой его несовместимость с одним из примером? Такая ситуация может проявиться в одном из двух описанных ниже вариантов.

- Некоторый пример может оказаться ☞ **ложно отрицательным** для данной гипотезы, если гипотеза утверждает, что он должен быть отрицательным, но фактически этот пример положителен. В частности, новый пример  $X_{13}$ , описанный выражением

$$\text{Patrons}(X_{13}, \text{Full}) \wedge \text{Wait}(X_{13}, 0-10) \wedge \neg \text{Hungry}(X_{13}) \wedge \dots \wedge \text{WillWait}(X_{13})$$

был бы ложно отрицателен для гипотезы  $H_r$ , приведенной выше. Из гипотезы  $H_r$  и описания этого можем вывести и выражение  $\text{WillWait}(X_{13})$ , которое как раз является утверждением данного примера, и выражение  $\neg \text{WillWait}(X_{13})$ , которое представляет собой предсказание самой гипотезы. Поэтому в данном случае гипотеза и пример логически несовместимы.

- Пример может быть ☞ **ложно положительным** для данной гипотезы, если в гипотезе утверждается, что он должен быть положительным, но фактически он является отрицательным<sup>1</sup>.

Если некоторый пример является ложно положительным или ложно отрицательным применительно к некоторой гипотезе, то данный пример и данная гипотеза являются логически несовместимыми друг с другом. При условии, что рассматриваемый пример представляет собой правильное наблюдение факта, такая ситуация позволяет исключить данную гипотезу. С точки зрения логики соответствующая операция исключения гипотезы полностью аналогична операции применения пра-

<sup>1</sup> Термины “ложно положительный” и “ложно отрицательный” используются также в медицине для описания ошибочных результатов лабораторных исследований. Результат исследования является ложно положительным, если он указывает, что у пациента имеется диагностируемое заболевание, тогда как в действительности у пациента этого заболевания нет.

вила резолюции в логическом выводе (см. главу 9); в этой аналогии дизъюнкция гипотез соответствует выражению, а пример соответствует литералу, который взаимно уничтожается с одним из литералов выражения. Поэтому в принципе может быть обеспечено обучение на примерах обычной системы логического вывода путем удаления одной или нескольких гипотез. В частности, предположим, что пример оформлен в виде высказывания  $I_1$ , а пространство гипотез представляет собой высказывание  $H_1 \vee H_2 \vee H_3 \vee H_4$ . В таком случае, если высказывание  $I_1$  несовместимо с выражениями  $H_2$  и  $H_3$ , то система логического вывода может сформировать новое высказывание  $H_1 \vee H_4$ , соответствующее уточненному пространству гипотез.

Таким образом индуктивное обучение в логической постановке задачи можно охарактеризовать как процесс постепенного устранения гипотез, несовместимых с примерами, и сужения пространства возможных гипотез. Но поскольку пространство гипотез обычно является колоссальным (или даже бесконечным, в случае логики первого порядка), не рекомендуется даже пытаться создать систему обучения с использованием доказательства теорем на основе резолюции и полного перебора пространства гипотез. Вместо этого в этой главе будут описаны два подхода, которые позволяют находить логически совместимые гипотезы с гораздо меньшими усилиями.

### Поиск текущей наилучшей гипотезы

В основе метода поиска  $\triangleright$  текущей наилучшей гипотезы лежит подход, предусматривающий сопровождение единственной гипотезы и ее корректировку по мере поступления новых примеров в целях поддержки совместимости. Основной алгоритм этого метода был впервые описан Джоном Стюартом Миллом [1049], но вполне мог быть изобретен еще раньше.

Предположим, что в нашем распоряжении имеется определенная гипотеза, скажем  $H_r$ , которая нас полностью устраивает. До тех пор пока каждый новый пример остается с ней совместимым, не нужно ничего делать. Наконец, вслед за ними поступает ложно отрицательный пример,  $X_{13}$ . Что теперь следует делать? На рис. 19.1, *a* гипотеза  $H_r$  показана схематически в виде области; все, что находится внутри прямоугольника, входит в состав расширения  $H_r$ . Примеры, которые фактически встретились до сих пор, показаны как “+” или “-”, и рисунок наглядно демонстрирует, что гипотеза  $H_r$  правильно классифицирует все примеры как положительные или отрицательные примеры предиката *WillWait*. Показанный на рис. 19.1, *b* новый пример (обозначенный кружком) является ложно отрицательным — в гипотезе утверждается, что он должен быть отрицательным, но фактически этот пример положителен. Расширение гипотезы должно быть увеличено с целью его включения. Такая операция называется  $\triangleright$  обобщением; одно из возможных обобщений показано на рис. 19.1, *c*. После этого на рис. 19.1, *d* показан ложно положительный пример — в гипотезе утверждается, что новый пример (обозначенный кружком) должен быть положительным, но фактически этот пример отрицателен. Расширение гипотезы должно быть уменьшено в целях исключения данного примера. Такая операция называется  $\triangleright$  уточнением; на рис. 19.1, *e* показано одно из возможных уточнений данной гипотезы. Отношения “более общий чем” и “более конкретный чем” между гипотезами позволяют создать пространство гипотез с такой логической структурой, которая дает возможность осуществлять эффективный поиск.

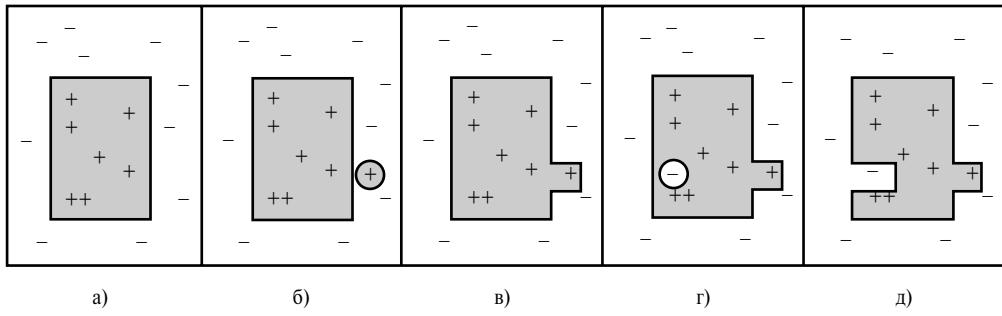


Рис. 19.1. Операции уточнения и обобщения: совместимая гипотеза (а); ложно отрицательный пример (б); гипотеза обобщена (в); ложно положительный пример (г); гипотеза уточнена (д)

Теперь можно определить алгоритм Current-Best-Learning, приведенный в листинге 19.1. Обратите внимание на то, что каждый раз при рассмотрении возможности обобщения или уточнения гипотезы необходимо проверять согласованность этих операций с другими примерами, поскольку произвольное увеличение/уменьшение расширения может привести к включению/исключению рассматривавшихся ранее отрицательных/положительных примеров.

**Листинг 19.1. Алгоритм обучения по методу поиска текущей наилучшей гипотезы.** Он осуществляет поиск совместимой гипотезы и выполняет возврат, если не удается найти какое-либо совместимое уточнение/обобщение

---

```

function Current-Best-Learning(examples) returns гипотеза
    H  $\leftarrow$  любая гипотеза, совместимая с первым примером в множестве
          примеров examples
    for each из оставшихся примеров в множестве examples do
        if пример e является ложно положительным
            применительно к H then
                H  $\leftarrow$  выбрать уточнение H, совместимое
                      с примерами examples
        else if пример e является ложно положительным
            применительно к H then
                H  $\leftarrow$  выбрать обобщение H, совместимое
                      с примерами examples
        if не удается найти какое-либо совместимое
            уточнение/обобщение then неудачное завершение поиска
    return H

```

---

Выше в данном разделе обобщение и уточнение были определены как операции, модифицирующие расширение гипотезы. Теперь необходимо точно определить, как эти операции могут быть реализованы в виде синтаксических операций, которые вносят изменения в потенциальное определение, связанное с гипотезой, чтобы эти операции можно было выполнять с помощью какой-то программы. Прежде чем приступить к решению этой задачи, необходимо отметить, что обобщение и уточнение также представляют собой логические связи между гипотезами. Если гипотеза  $H_1$  с определением  $C_1$  является обобщением гипотезы  $H_2$  с определением  $C_2$ , то должно иметь место следующее выражение:

$$\forall x \ C_2(x) \Rightarrow C_1(x)$$

Поэтому, чтобы сформировать обобщение гипотезы  $H_2$ , необходимо просто найти определение  $C_1$ , которое логически следует из  $C_2$ . Такая задача решается довольно легко. Например, если  $C_2(x)$  представляет собой высказывание  $\text{Alternate}(x) \wedge \text{Patrons}(x, \text{Some})$ , то одно из возможных обобщений задается в виде  $C_1(x) \equiv \text{Patrons}(x, \text{Some})$ . Такая форма операции обобщения называется **удалением условий**. Интуитивно ясно, что в результате такой операции формируется более слабое определение и поэтому гипотеза допускает использование более крупного множества положительных примеров. Может быть также предусмотрен целый ряд других операций обобщения, в зависимости от языка, в котором выражаются эти операции. Аналогичным образом, уточнение гипотезы может осуществляться путем введения дополнительных условий в ее потенциальное определение или путем удаления дизъюнктов из какого-то дизъюнктивного определения. Рассмотрим, как могут быть выполнены подобные операции в задаче с рестораном, используя данные, приведенные в табл. 18.1.

- Первый пример,  $X_1$ , является положительным. Выражение  $\text{Alternate}(X_1)$  имеет истинное значение, поэтому допустим, что начальная гипотеза имеет такой вид:

$$H_1: \forall x \text{WillWait}(x) \Leftrightarrow \text{Alternate}(x)$$

- Второй пример,  $X_2$ , отрицателен. Гипотеза  $H_1$  предсказывает, что он должен быть положительным, поэтому данный пример — должно положителен. Это означает, что необходимо уточнить гипотезу  $H_1$ . Это можно сделать, введя дополнительное условие, позволяющее исключить пример  $X_2$ . Один из возможных вариантов состоит в следующем:

$$H_2: \forall x \text{WillWait}(x) \Leftrightarrow \text{Alternate}(x) \wedge \text{Patrons}(x, \text{Some})$$

- Третий пример,  $X_3$ , является положительным. Гипотеза  $H_2$  предсказывает, что он должен быть отрицательным, поэтому данный пример — должно отрицательный. Это означает, что необходимо обобщить гипотезу  $H_2$ . Удалим условие  $\text{Alternate}$  и получим следующее:

$$H_3: \forall x \text{WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some})$$

- Четвертый пример,  $X_4$ , также положителен. Гипотеза  $H_3$  предсказывает, что он должен быть отрицательным, поэтому данный пример — должно отрицательный. Это означает, что необходимо обобщить гипотезу  $H_3$ . Условие  $\text{Patrons}$  удалить нельзя, поскольку такая операция приведет к созданию всеохватывающей гипотезы, которая будет несовместимой с примером  $X_2$ . Одна из возможностей состоит в добавлении дизъюнкта:

$$H_4: \forall x \text{WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some}) \\ \vee (\text{Patrons}(x, \text{Full}) \wedge \text{Fri/Sat}(x))$$

Итак, гипотеза уже начинает выглядеть как приемлемая. Очевидно, что есть и другие варианты, совместимые с первыми четырьмя примерами; ниже приведены два из них.

$$H_4': \forall x \text{WillWait}(x) \Leftrightarrow \neg \text{WaitEstimate}(x, 30-60) \\ H_4'': \forall x \text{WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some}) \\ \vee (\text{Patrons}(x, \text{Full}) \wedge \text{WaitEstimate}(x, 10-30))$$

Алгоритм Current-Best-Learning описан здесь в недетерминированной форме, поскольку в любой момент может существовать возможность применить несколько вариантов уточнения или обобщения. Выбранный вариант не всегда обеспечивает получение простейшей гипотезы и может привести также к безвыходной ситуации, в которой ни одна простая модификация гипотезы не будет совместимой со всеми данными. В подобных случаях программа должна выполнить возврат к предыдущей точке выбора.

Алгоритм Current-Best-Learning и его варианты использовались во многих системах машинного обучения, начиная с программы “обучения распознаванию арок” Патрика Уинстона [1602]. Однако при большом количестве экземпляров примеров и большом пространстве гипотез возникают некоторые описанные ниже трудности.

1. В алгоритме снова и снова проводится проверка всех предыдущих экземпляров при каждой модификации, а это требует больших затрат.
2. Процесс поиска может быть связан с очень интенсивным перебором с возвратами. Как было показано в главе 18, размеры пространства гипотез могут определяться двойной экспоненциальной зависимостью.

### Поиск на основе оценки наименьшего вклада

Перебор с возвратами возникает из-за того, что в подходе, предусматривающем поиск текущей наилучшей гипотезы, приходится выбирать конкретную гипотезу как наилучшее текущее предположение, даже несмотря на то, что еще отсутствует достаточный объем данных, позволяющий убедиться в правильности этого выбора. Вместо этого можно было бы просто “держать под рукой” те и только те гипотезы, которые являются совместимыми со всеми поступившими до сих пор данными. В таком случае каждый новый экземпляр данных либо не оказывает никакого влияния на состав гипотез, либо позволяет избавиться от некоторых гипотез. Напомним, что первоначальное пространство гипотез может рассматриваться как следующее дизъюнктивное высказывание:

$$H_1 \vee H_2 \vee H_3 \vee \dots \vee H_n$$

По мере обнаружения того, что различные гипотезы являются несовместимыми с примерами, эта дизъюнкция сужается и остаются только те гипотезы, которые еще не исключены. Это означает, что при условии, что первоначальное пространство гипотез действительно содержит правильный ответ, этот правильный ответ должен также содержаться в сокращенной дизъюнкции, поскольку были удалены только неправильные гипотезы. Множество оставшихся гипотез называется **пространством версий**, а соответствующий алгоритм обучения (приведенный в листинге 19.2) называется алгоритмом обучения в пространстве версий (а также алгоритмом **удаления потенциальных гипотез**).

**Листинг 19.2. Алгоритм обучения в пространстве версий. Он находит подмножество гипотез  $V$ , совместимое с примерами `examples`**

---

```
function Version-Space-Learning(examples) returns пространство версий
    local variables:  $V$ , пространство версий – множество всех гипотез
```

$V \leftarrow$  множество всех гипотез

---

```

for each из примеров  $e$  в множестве примеров  $\text{examples}$  do
    if множество  $V$  не пусто then  $V \leftarrow \text{Version-Space-Update}(V, e)$ 
    return  $V$ 

function Version-Space-Update( $V, e$ ) returns обновленное
    пространство версий
         $V \leftarrow \{h \in V: \text{гипотеза } h \text{ совместима с } e\}$ 

```

---

Одним важным свойством этого алгоритма является то, что он инкрементный, — при его использовании никогда не приходится возвращаться и повторно исследовать старые примеры, поскольку в любом случае гарантируется, что оставшиеся гипотезы остаются совместимыми с ними. Этот алгоритм также представляет собой алгоритм поиска **с наименьшим вкладом**, поскольку в нем не используются произвольные варианты выбора (сравните его с алгоритмом планирования с частичным упорядочением, который приведен в главе 11). Но при использовании этого алгоритма возникает одна очевидная проблема. Как уже было сказано, пространство гипотез часто имеет огромные размеры; как же в таком случае можно записать соответствующую ему огромную дизьюнкцию?

Для анализа этой ситуации можно применить следующую очень полезную простую аналогию. Как представить все действительные числа в интервале между 1 и 2? В конце концов, ведь количество этих чисел бесконечно велико! Ответ на этот вопрос состоит в том, что должно использоваться интервальное представление, в котором задаются границы этого множества,  $[1, 2]$ . Это представление применимо в связи с тем, что во множестве действительных чисел задано упорядочение.

В пространстве версий также существует отношение упорядочения, а именно отношение обобщения/уточнения. Но это — отношение частичного упорядочения, поэтому каждая граница пространства представляет собой не точку, а скорее множество гипотез, называемое **граничным множеством**. Превосходной особенностью подхода, основанного на понятии граничного множества, является то, что он позволяет представить все пространство версий с использованием только двух граничных множеств — наиболее общего граничного множества (**G-множества**, где G — сокращение от general) и наиболее конкретного граничного множества (**S-множества**, где S — сокращение от specific). При этом гарантируется совместимость с примерами всех версий, которые находятся в пределах между этими двумя множествами. Прежде чем перейти к доказательству этого утверждения, подытожим сказанное выше в настоящей главе.

- Текущее пространство версий представляет собой множество гипотез, совместимых со всеми примерами, встретившимися до сих пор. Это пространство представлено с помощью S- и G-множества, каждое из которых представляет собой множество гипотез.
- Каждый элемент S-множества является совместимым со всеми полученными до сих пор результатами наблюдений, и нет ни одной совместимой гипотезы, которая была бы более конкретной, чем этот элемент.
- Каждый элемент G-множества является совместимым со всеми полученными до сих пор результатами наблюдений, и нет ни одной совместимой гипотезы, которая была бы более общей, чем этот элемент.

Необходимо, чтобы первоначальное пространство версий (существовавшее до того, как встретились какие-либо примеры) содержало все возможные гипотезы. Для этого достаточно задать G-множество таким образом, чтобы оно содержало гипотезу *True* (гипотезу, которая включает все возможные случаи), а S-множество — так, чтобы оно содержало гипотезу *False* (гипотезу, расширение которой пусто).

На рис. 19.2 показана общая структура представления пространства версий с помощью граничных множеств. Для того чтобы показать, что это представление является достаточным, необходимо определить два описанных ниже свойства.

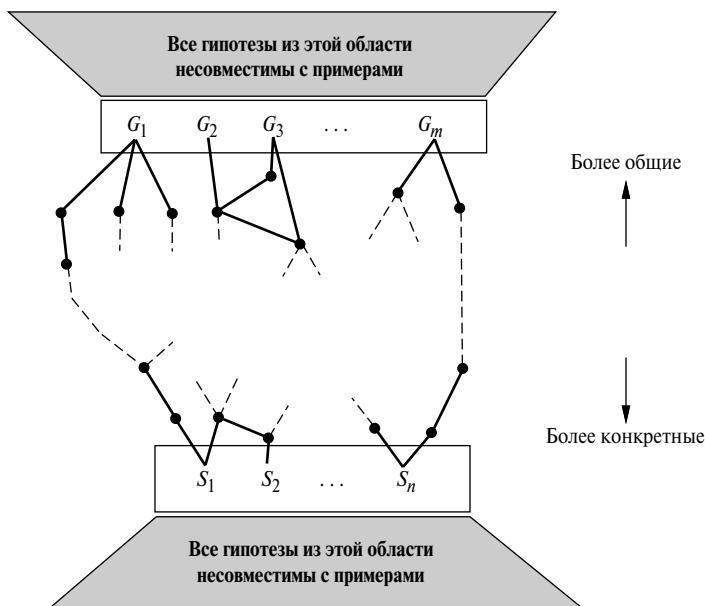
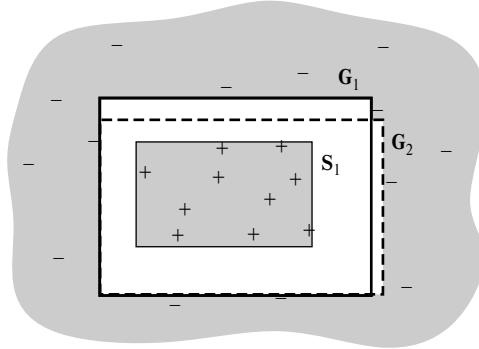


Рис. 19.2. Схематическая иллюстрация того, что пространство версий содержит все гипотезы, совместимые с примерами

1. Каждая совместимая гипотеза (отличная от гипотез граничных множеств) является более конкретной, чем некоторый элемент G-множества, и более общей, чем некоторый элемент S-множества (это означает, что за рамками пространства версий не остается ни одной “забытой гипотезы”). Такое свойство следует непосредственно из определений S- и G-множеств. Если бы существовала какая-то не охваченная гипотеза  $h$ , то она была бы не более конкретной, чем любой элемент G-множества, поскольку в таком случае она принадлежала бы к G-множеству, и не более общей, любой элемент S-множества, поскольку в таком случае она принадлежала бы к S-множеству.
2. Любая гипотеза, более конкретная, чем некоторый элемент G-множества, и более общая, чем некоторый элемент S-множества, является совместимой гипотезой. (Это означает, что между границами нет “пустот”, в которых находились бы гипотезы, не охваченные отношениями обобщения/уточнения.) Любая гипотеза  $h$ , лежащая между S- и G-множествами, должна отвергать все отрицательные примеры, отвергнутые каждым элементом G-множества (поскольку она является более конкретной), и должна принимать все положительные примеры, принятые каждым элементом S-множества (поскольку она является более общей).

жительные примеры, принятые любым элементом  $S$ -множества (поскольку она является более общей). Таким образом, гипотеза  $h$  должна согласовываться со всеми примерами и поэтому не может быть несовместимой. Соответствующая ситуация показана на рис. 19.3 — не существует каких-либо известных примеров вне  $S$ -множества, но внутри  $G$ -множества, поэтому любая гипотеза в промежутке между ними должна быть совместимой.



*Рис. 19.3. Расширения  $G$ - и  $S$ -множества, показанные в виде отдельных элементов. Между двумя множествами границ не находится ни один известный пример*

Таким образом, было показано, что если будет обеспечено правильное сопровождение  $S$ - и  $G$ -множества в соответствии с их определениями, то эти множества обеспечивают удовлетворительное представление пространства версий. Остается единственная проблема — как обновлять  $S$ - и  $G$ -множество с учетом нового примера (выполнение этой операции возложено на функцию *Version-Space-Update*). На первый взгляд эта задача может показаться довольно затруднительной, но на основе определений множеств и с помощью рис. 19.2 не слишком сложно составить необходимый для этого алгоритм.

Необходимо прежде всего обеспечить использование элементов  $S_i$  и  $G_i$  из  $S$ - и  $G$ -множества соответственно. Применительно к каждому из таких элементов каждый новый экземпляр примера может быть либо положительным или либо отрицательным. Возникающие при этом варианты описаны ниже.

1. Ложно положительный пример для  $S_i$ . Появление такого примера означает, что гипотеза  $S_i$  является слишком общей, но (по определению) совместимое уточнение для гипотезы  $S_i$  отсутствует, поэтому ее необходимо исключить из  $S$ -множества.
2. Ложно отрицательный пример для  $S_i$ . Появление такого примера означает, что гипотеза  $S_i$  является слишком конкретной, поэтому она заменяется всеми своими непосредственными обобщениями, при соблюдении условия, что они остаются более конкретными, чем любой элемент  $G$ -множества.
3. Ложно положительный пример для  $G_i$ . Появление такого примера означает, что гипотеза  $G_i$  является слишком общей, поэтому она заменяется всеми

своими непосредственными уточнениями, при соблюдении условия, что они остаются более общими, чем любой элемент S-множества.

4. Ложно отрицательный пример для  $G_i$ . Появление такого примера означает, что гипотеза  $G_i$  является слишком конкретной, но (по определению) совместимое обобщение для гипотезы  $G_i$  отсутствует, поэтому ее необходимо исключить из G-множества.

Указанные выше операции продолжаются применительно к каждому новому экземпляру примера до тех пора, пока не произойдет одно из перечисленных ниже событий.

1. В пространстве версий останется одно и только одно определение некоторой концепции, и в этом случае оно будет возвращено как уникальная гипотеза.
2. Пространство версий свернется (либо S-, либо G-множество станет пустым), и это будет означать, что для данного обучающего множества не существуют совместимые гипотезы. Такая ситуация аналогична неудачному завершению поиска, которое возникает в простом варианте алгоритма формирования дерева решений.
3. Примеры закончатся, а в пространстве версий останется несколько гипотез. Это означает, что пространство версий представляет собой дизъюнкцию гипотез. Если при поступлении любого нового примера все дизъюнкты этой дизъюнкции согласуются, то можно возвратить определяемую ими классификацию данного примера. Если же они не согласуются, один из вариантов состоит в использовании мажоритарного голосования.

Рекомендуем читателю в качестве упражнения попытаться применить алгоритм Version-Space-Learning к данным о ресторане.

Описанный здесь подход к использованию пространства версий характеризуется описанными ниже принципиальными недостатками.

- Если проблемная область характеризуется наличием шума или недостаточного количества атрибутов для точной классификации, то пространство версий всегда свертывается.
- Если в пространстве гипотез допускается наличие неограниченного количества дизъюнкций, то S-множество всегда будет содержать единственную наиболее конкретную гипотезу, а именно дизъюнкцию описаний положительных примеров, встретившихся до сих пор. Аналогичным образом, G-множество будет содержать просто отрицание дизъюнкции описаний отрицательных примеров.
- Для некоторых пространств гипотез количество элементов в S- или в G-множестве может увеличиваться в экспоненциальной зависимости от количества атрибутов, что не позволяет решить задачу обучения, даже несмотря на то, что для этих пространств гипотез существуют эффективные алгоритмы обучения.

Полностью эффективное решение проблемы шума не найдено до сих пор. Проблемы дизъюнкции можно решить, допуская использование ограниченных форм дизъюнкции или включая ~~иерархию обобщения~~ более общих предикатов. Например, вместо применения дизъюнкции  $\text{WaitEstimate}(x, 30-60) \vee \text{WaitEstimate}(x, >60)$  можно

ввести в действие единственный литерал  $\text{LongWait}(x)$ . Для реализации такого подхода можно легко дополнить множество операций обобщения и уточнения.

Рассматриваемый в этом разделе чистый алгоритм сопровождения пространства версий был впервые применен в системе Meta-Dendral, которая была предназначена для изучения правил предсказания того, как молекулы разделяются на фрагменты в массовом спектрометре [202]. Система Meta-Dendral показала свою способность вырабатывать правила, которые оказались достаточно новаторскими, чтобы гарантировать их публикацию в журнале аналитической химии; это были первые реальные научные знания, полученные с помощью компьютерной программы. Такой алгоритм использовался также в изящной системе Lex [1066], которая проявила способность к обучению методам решения задач символического интегрирования, анализируя свои собственные удачи и неудачи. Хотя методы сопровождения пространства версий, по-видимому, не могут найти практического применения в большинстве реальных задач обучения, в основном из-за проблемы шума, они позволяют многое понять в отношении того, какова логическая структура пространства гипотез.

## 19.2. ПРИМЕНЕНИЕ ЗНАНИЙ В ОБУЧЕНИИ

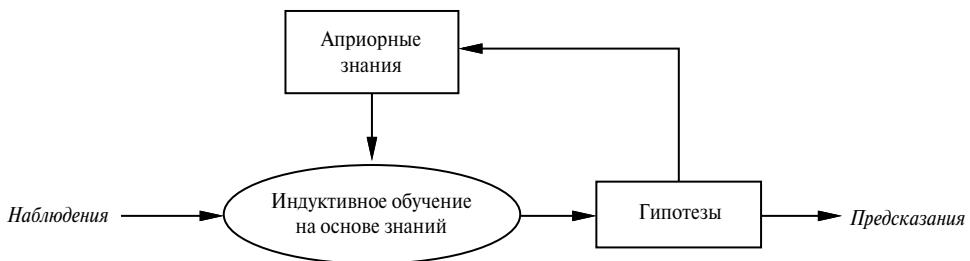
В предыдущем разделе описывалась простейшая постановка задачи индуктивного обучения. А в этом разделе речь пойдет о логических связях между гипотезами, описаниями примеров и классификациями, что позволяет понять роль априорных знаний. Допустим, что *Descriptions* обозначает конъюнкцию всех описаний примеров в обучающем множестве, а *Classifications* — конъюнкцию всех классификаций примеров. В таком случае гипотеза, которая позволяет “объяснить результаты наблюдений”, должна удовлетворять следующему свойству (напомним, что  $\models$  означает “влечет за собой логически”):

$$\text{Hypothesis} \wedge \text{Descriptions} \models \text{Classifications} \quad (19.3)$$

Мы будем называть связь такого рода ~~ограничением логического следствия~~, в котором *Hypothesis* представляет собой “неизвестное”. Чисто индуктивное обучение сводится к разрешению этого ограничения, притом что гипотеза *Hypothesis* извлекается из некоторого заранее определенного пространства гипотез. Например, если дерево решений рассматривается как логическая формула (см. уравнение 19.1), то дерево решений, совместимое со всеми примерами, будет удовлетворять уравнению 19.3. Если же на логическую форму гипотезы не налагаются никакие ограничения, то, разумеется, условие *Hypothesis*=*Classifications* также удовлетворяет этому ограничению. Согласно принципу бритвы Оккама, следует отдавать предпочтение небольшим, совместимым гипотезам, поэтому мы должны попытаться найти лучшее решение, чем просто запоминание примеров.

Такой простой подход к индуктивному обучению, в котором не предусматривалось использование знаний, господствовал в искусственном интеллекте до начала 1980-х годов. А современный подход состоит в том, что должны проектироваться агенты, которые ~~уже кое-что знают~~ и пытаются освоить в процессе обучения некоторую дополнительную информацию. На первый взгляд открытие, приведшее к такой смене подходов, не кажется чрезвычайно глубоким, но оно стало причиной

весьма существенного изменения способа проектирования агентов. Кроме того, это открытие может иметь определенное отношение к рассматриваемым в данной книге теориям о том, как развивается сама наука. Общая идея этого открытия показана на рис. 19.4.



*Рис. 19.4. Схема, которая показывает, как в процессе обучения с накоплением знаний используется и со временем обогащается запас фоновых знаний*

Если должен быть создан автономный обучающийся агент, который использует фоновые знания, то в этом агенте необходимо предусмотреть некоторый метод получения в первую очередь фоновых знаний, для того, чтобы их можно было использовать в новых эпизодах обучения. Применяемый при этом метод сам должен представлять собой процесс обучения. Поэтому история существования агента должна быть таковой, чтобы ее можно было охарактеризовать как поступательное, или инкрементное, развитие. Разумеется, что агент может начать свое существование без какого-либо запаса знаний, осуществляя индуктивные выводы в условиях отсутствия знаний, как и добрая старая программа чисто индуктивного вывода. Но как только агент съест плод с Древа познания, он не будет больше обязан заниматься подобными примитивными рассуждениями, а должен использовать свои фоновые знания для все более эффективного обучения. Таким образом, вопрос заключается в том, как фактически следует использовать знания в обучении.

### Некоторые простые примеры

Рассмотрим некоторые общезвестные примеры обучения на основе фоновых знаний. При этом необходимо учитывать, что во многих формах поведения, требующих логического вывода релевантных гипотез в связи с поступлением новых результатов наблюдения, не применимы даже самые элементарные принципы чистой индукции.

- Иногда возникают ситуации, в которых люди сразу же приходят к общим заключениям после одного лишь наблюдения. Перу Гэри Ларсона принадлежит юмористический рисунок, на котором вооруженный очками пещерный человек Зог поджаривает на огне ящерицу, насаженную на заостренную палочку. За ним наблюдает восхищенная толпа его менее интеллектуальных соплеменников, которые до сих пор всегда разогревали свою пищу над огнем, держа ее голыми руками. Этот наглядный пример является достаточным для того, чтобы ознакомить зрителей с общим принципом безболезненного приготовления пищи.
- В качестве еще одного примера рассмотрим случай, в котором путешественник, прибывший в Бразилию, встречает первого в своей жизни бразильца. Ус-

lyшав от него речь на португальском языке, путешественник сразу же приходит к выводу, что бразильцы говорят на португальском, но узнав, что его собеседника зовут Фернандо, он не делает вывод, что все бразильцы носят имя Фернандо. Аналогичные примеры можно найти в любой научной области. Например, когда начинающий студент-физик измеряет плотность и проводимость медного образца при определенной температуре, он без колебаний обобщает эти данные на все предметы из меди. Но измерив затем массу этого образца, студент даже не рассматривает такую гипотезу, что все предметы из меди имеют такую же массу. С другой стороны, он может вполне резонно сделать такое обобщение относительно всех монет одинакового достоинства.

- Наконец, рассмотрим случай, в котором незнакомый с фармакологией, но глубоко освоивший специальность диагностики студент-медик присутствует на консультации, которую дает пациенту опытный терапевт. После ряда вопросов и ответов специалист сообщает пациенту, чтобы он прошел курс лечения каким-то конкретным антибиотиком. На основании увиденного студент-медик выводит общее правило, что данный конкретный антибиотик является эффективным средством лечения при данном конкретном типе инфекции.

Все эти случаи относятся к такому типу, в котором  $\text{Background}$  *использование фоновых знаний обеспечивает гораздо более быстрое обучение по сравнению с тем, чего можно ожидать при использовании программы чисто индуктивного обучения.*

### Некоторые общие схемы

В каждом из предыдущих примеров имеется возможность обратиться к априорным знаниям, чтобы попытаться обосновать выбранный способ обобщения. В данном разделе рассмотрим, какого рода ограничения логического следствия применяются в каждом из этих случаев. В таких ограничениях, кроме гипотезы *Hypothesis*, описаний наблюдаемых примеров *Descriptions* и классификации *Classifications*, применяются фоновые знания *Background*.

В случае поджаривания ящерицы пещерный человек проводит обобщение, объясняя своим соплеменникам успех, достигнутый с помощью заостренной палочки, — он держит ящерицу над огнем и вместе с тем не приближает руки к огню. Из этого объяснения его соплеменники могут вывести общее правило, что для поджаривания небольших мясистых съедобных объектов можно использовать любой длинный, твердый, заостренный предмет. Процесс обобщения такого рода получил название  $\text{Explaination-based Learning}$ , или сокращенно **EBL** (Explanation-Based Learning). Обратите внимание на то, что общее правило следует логически из фоновых знаний, которыми обладают рассматриваемые в данном примере пещерные люди. Поэтому ограничения логического следствия, которые удовлетворяются в процессе EBL, являются таковыми:

$$\begin{aligned} \text{Hypothesis} \wedge \text{Descriptions} &\models \text{Classifications} \\ \text{Background} &\models \text{Hypothesis} \end{aligned}$$

Поскольку в процессе EBL используется уравнение 19.3, на первых порах он рассматривался как наилучший способ обучения на примерах. Но в связи с тем, что для обучения на основе объяснения требуется, чтобы фоновые знания были достаточными для объяснения гипотезы *Hypothesis*, которая в свою очередь объясняет ре-

зультаты наблюдений, ~~в действительности агент из полученного экземпляра примера не извлекает никаких принципиально новых знаний~~. Дело в том, что агент мог бы просто вывести логическим путем предъявленный ему пример из того, что он уже знает, хотя для этого, возможно, потребовался бы неоправданно большой объем вычислений. В настоящее время обучение на основе объяснения рассматривается как метод преобразования теорий, складывающихся из основных принципов, в полезные знания специального назначения. Алгоритмы обучения на основе объяснения рассматриваются в разделе 19.3.

В следующем примере, когда речь шла о путешественнике в Бразилию, ситуация совсем иная, поскольку не каждый путешественник знает о том, почему встретившийся ему бразилец говорит именно на португальском языке, если этот путешественник никогда не слышал о соответствующих буллах, изданных папой Римским. Более того, к аналогичному обобщению мог бы прийти путешественник, полностью незнакомый с историей португальских колоний. Соответствующие априорные знания состоят в том, что в каждой конкретной стране большинство людей, как правило, говорят на одном и тот же языке; с другой стороны, не предполагается, что имя Фернандо носят все бразильцы, поскольку указанная закономерность, касающаяся общего языка для всей страны, не распространяется на имена. Аналогичным образом, начинающий студент-физик вряд ли мог бы объяснить, почему измеренные им значения проводимости и плотности меди являются именно таковыми. Но он знает, что проводимость объекта зависит и от материала, из которого состоит объект, и от его температуры. В данном случае априорные знания *Background* касаются ~~релевантности~~ множества характеристик по отношению к целевому предикату. Эти знания, наряду с результатами наблюдений, позволяют агенту вывести новое, общее правило, которое объясняет результаты наблюдений, следующим образом:

$$\begin{aligned} \text{Hypothesis} \wedge \text{Descriptions} &\models \text{Classifications} \\ \text{Background} \wedge \text{Descriptions} \wedge \text{Classifications} &\models \text{Hypothesis} \end{aligned} \quad (19.4)$$

Мы будем называть обобщение такого рода ~~обучением с учетом релевантности~~, или сокращенно **RBL** (Relevance-Based Learning), хотя такое название указанного метода обучения не является общепринятым. Обратите внимание на то, что в обучении с учетом релевантности используются результаты наблюдений, но не вырабатываются гипотезы, выходящие за пределы логического содержания фоновых знаний и наблюдений. Оно представляет собой дедуктивную форму обучения и поэтому не может само по себе рассматриваться как создание новых знаний с пустого места.

А в том случае, когда студент-медик наблюдает за работой опытного врача, предполагается, что априорных знаний студента достаточно, чтобы прийти к заключению на основании симптомов о том, каково заболевание *D* этого пациента. Но этого не достаточно, чтобы объяснить тот факт, что врач прописал конкретное лекарство *M*. Студент должен выдвинуть гипотезу, касающуюся еще одного правила, а именно, что лекарство *M* обычно является эффективным средством против заболевания *D*. С учетом этого правила и априорных знаний студент может теперь объяснить, почему в данном конкретном случае опытный врач прописал лекарство *M*. Этот пример можно обобщить так: указать, что он основан на приведенном ниже ограничении логического следствия.

$$\text{Background} \wedge \text{Hypothesis} \wedge \text{Descriptions} \models \text{Classifications} \quad (19.5)$$

Это означает, что для объяснения примеров объединяются фоновые знания и новая гипотеза. Как и при чисто индуктивном обучении, алгоритм обучения должен выдвигать гипотезы, которые являются как можно более простыми и совместимыми с данным ограничением. Алгоритмы, удовлетворяющие ограничению 19.5, называются алгоритмами **индуктивного обучения на основе знаний**, или сокращенно **KBIL** (Knowledge-Based Inductive Learning).

Алгоритмы KBIL, которые подробно описаны в разделе 19.5, в основном исследовались в области **индуктивного логического программирования**, или сокращенно **ILP** (Inductive Logic Programming). В системах ILP априорные знания выполняют две описанные ниже ключевые функции при решении задачи уменьшения сложности обучения.

1. Поскольку любая сформированная гипотеза должна быть совместимой с априорными знаниями, а также с новыми наблюдениями, эффективный размер пространства гипотез сокращается таким образом, чтобы в него были включены только теории, согласованные с тем, что уже известно.
2. Для любого конкретного множества наблюдений размер гипотезы, требуемый для формирования объяснения полученных результатов наблюдений, может быть намного сокращен, поскольку доступны априорные знания, на которые могут опираться новые правила при формировании объяснений результатов наблюдений. А чем короче гипотеза, тем проще ее сформулировать.

Системы ILP не только позволяют использовать в процессе индукции априорные знания, но и дают возможность формулировать гипотезы на языке общей логики первого порядка, а не на ограниченном языке, основанном на атрибуатах, который рассматривается в главе 18. Это означает, что подобные системы могут осуществлять обучение в таких вариантах среды, которые остаются недоступными для понимания при использовании более простых систем.

### 19.3. ОБУЧЕНИЕ НА ОСНОВЕ ОБЪЯСНЕНИЯ

Как было описано во введении в данной главе, обучение на основе объяснения представляет собой метод извлечения общих правил из отдельных результатов наблюдений. В качестве примера рассмотрим задачу дифференцирования и упрощения алгебраических выражений (упр. 9.15). После выполнения дифференцирования по  $X$  такого выражения, как  $X^2$ , будет получено выражение  $2X$ . (Обратите внимание на то, что мы используем прописную букву для обозначения арифметического неизвестного  $X$ , чтобы отличить его от логической переменной  $x$ .) В системе формирования логических рассуждений такая цель может быть выражена как `Ask(Derivative(X2, X)=d, KB)` с предполагаемым решением  $d=2X$ .

Любой, кто знает дифференциальное исчисление, может найти данное решение, лишь “ознакомившись с условиями задачи”, благодаря приобретенным ранее навыкам решения подобных задач. А перед студентом, столкнувшимся с подобными задачами впервые, или перед программой, не накопившей опыта в решении этих задач, возникают гораздо более существенные затруднения. Применение стандартных правил дифференцирования в конечном итоге приводит к получению выражения

$1 \times (2 \times (X^{(2-1)}))$ , а это выражение в дальнейшем упрощается до  $2X$ . В логической программной реализации, подготовленной авторами данной книги, для решения такой задачи потребовалось 136 шагов доказательства, причем 99 из этих шагов относились к тупиковым ветвям доказательства. Тот, кто накопил такой отрицательный опыт, вполне естественно, стремится к тому, чтобы та же программа решала аналогичную задачу намного более быстро, встретившись с ней в следующий раз.

В компьютерных науках для ускорения работы программ уже в течение долгого времени используется метод **запоминания** (memoization), который предусматривает сохранение в памяти результатов вычислений. Фундаментальная идея, лежащая в основе запоминающих функций, состоит в том, что должна накапливаться база данных, состоящая из входных/выходных пар; после вызова функции в первую очередь выполняется проверка базы данных для определения того, можно ли обойтись без решения рассматриваемой задачи с самого начала. Обучение на основе объяснения представляет собой значительный шаг вперед, поскольку в нем предусматривается создание общих правил, охватывающих целый класс вариантов. В случае дифференцирования метод запоминания позволяет записать в память информацию о том, что производная от  $X^2$  по  $X$  равна  $2X$ , но потребует от агента, чтобы он вычислял производную от  $Z^2$  по  $Z$  самого начала. Однако было бы желательно иметь возможность извлечь общее правило<sup>2</sup>, что для любого арифметического неизвестного  $u$  производная от  $u^2$  по  $u$  равна  $2u$ . В терминах логики такая мысль может быть выражена с помощью следующего правила:

$$\text{ArithmetiсUnknown}(u) \Rightarrow \text{Derivative}(u^2, u) = 2u$$

Если база знаний содержит такое правило, то для любого нового случая, который является экземпляром этого правила, может быть немедленно найдено решение.

Безусловно, это лишь тривиальный пример очень общего феномена. После того как достигнуто понимание какой-то идеи, она может быть обобщена и повторно использована в других обстоятельствах. Освоенная идея становится “очевидным” этапом мышления, после чего может использоваться в качестве строительного блока при решении еще более сложных задач. Альфреду Норту Уайтхеду [1583], в соавторстве с которым Берtrand Рассел написал свою знаменитую книгу *Principia Mathematica*, принадлежат такие слова: “Цивилизация развивается, увеличивая количество важных операций, которые ее представители могут выполнять, не задумываясь”. По-видимому, сам Уайтхед применял обучение на основе объяснения, чтобы истолковать полученные им результаты, по аналогии с тем, как Зог объяснял соплеменникам свое открытие. Если вы поняли основную идею примера с дифференцированием, то ваш мозг уже занимается извлечением из него основных принципов обучения на основе объяснения. Обратите внимание на то, что вам не требовалось заранее изобретать принципы обучения на основе объяснения, прежде чем вы ознакомились с этим примером. Как и пещерным людям, наблюдающим за манипуляциями Зога, вам (и авторам данной книги) нужен был пример для того, чтобы прийти к пониманию основных принципов. Именно поэтому часто гораздо проще сначала показать, почему так хороша какая-то идея, чем сразу же ее изложить.

---

<sup>2</sup> Безусловно, может быть также сформировано общее правило для  $u^n$ , но текущего примера достаточно, чтобы проиллюстрировать общий принцип.

## Извлечение общих правил из примеров

Обучение на основе объяснения базируется на том важном принципе, что вначале необходимо составить объяснение данного наблюдения с использованием априорных знаний, а затем сформировать определение класса случаев, для которого может использоваться такая же структура объяснения. Подобное определение может лежать в основу правила, охватывающего все случаи данного класса. Само “объяснение” может представлять собой логическое доказательство, но в более общем случае объяснением может служить любой процесс формирования рассуждений или решения задачи, этапы которого полностью определены. Ключом к успешному формированию объяснения является определение необходимых условий для того, чтобы такие же этапы процесса можно было применить к другому случаю.

Мы будем использовать в качестве рассматриваемой здесь системы формирования логических рассуждений простую систему доказательства теорем по методу обратного логического вывода, которая описана в главе 9. Дерево доказательства для случая  $\text{Derivative}(X^2, X) = 2X$  слишком велико для того, чтобы его можно было использовать в качестве примера, поэтому для иллюстрации применяемого метода обобщения рассмотрим более простую задачу. Предположим, что задача состоит в том, чтобы упростить выражение  $1 \times (0+X)$ . База знаний включает следующие правила:

```
Rewrite(u, v) ∧ Simplify(v, w) ⇒ Simplify(u, w)
Primitive(u) ⇒ Simplify(u, u)
ArithmeticUnknown(u) ⇒ Primitive(u)
Number(u) ⇒ Primitive(u)
Rewrite(1×u, u)
Rewrite(0+u, u)
...

```

Доказательство того, что ответом является  $X$ , приведено на рис. 19.5, сверху. В методе обучения на основе объяснения фактически одновременно формируются два дерева доказательства. Во втором дереве доказательства используется цель, описанная с помощью переменных, в которой константы первоначальной цели заменены переменными. По мере развития первоначального доказательства так же поэтапно развивается доказательство с помощью переменных, в котором применяются точно такие же приложения правил. Использование некоторых правил может вызвать конкретизацию определенных переменных. Например, для того чтобы можно было применить правило  $\text{Rewrite}(1×u, u)$ , необходимо вначале связать с 1 переменную  $x$  в подцели  $\text{Rewrite}(x×(y+z), v)$ . Аналогичным образом, переменная  $y$  должна быть связана с 0 в подцели  $\text{Rewrite}(y+z, v')$ , для того чтобы можно было использовать правило  $\text{Rewrite}(0+u, u)$ . После получения обобщенного дерева доказательства остается только взять листовые узлы (с необходимыми связываниями) и сформулировать общее правило для целевого предиката, следующим образом:

```
Rewrite(1 × (0+z), 0+z) ∧ Rewrite(0+z, z) ∧ ArithmeticUnknown(z)
⇒ Simplify(1×(0+z), z)
```

Обратите внимание на то, что первые два условия в левой части правила являются истинными независимо от значения  $z$ . Поэтому их можно удалить из данного правила, что приводит к получению такого соотношения:

```
ArithmeticUnknown(z) ⇒ Simplify(1×(0+z), z)
```

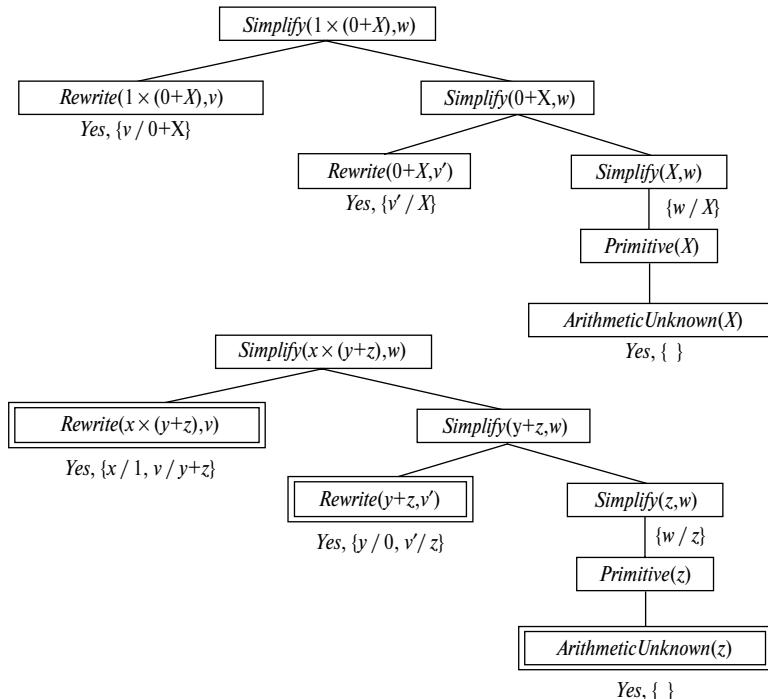


Рис. 19.5. Деревья доказательства для рассматриваемой задачи упрощения. Первое дерево демонстрирует доказательство для первоначального экземпляра задачи, на основании которого можно вывести следующее:

$ArithmeticUnknown(z) \Rightarrow Simplify(1 \times (0+z), z)$

Второе дерево демонстрирует доказательство для экземпляра задачи, в котором все константы заменены переменными, что позволяет вывести целый ряд других правил

Вообще говоря, из окончательного правила можно удалять такие условия, которые не налагают ограничений на переменные в правой части правила, поскольку при этом результирующее правило остается истинным и становится более эффективным. Обратите внимание на то, что условие  $ArithmeticUnknown(z)$  удалять нельзя, поскольку не все возможные значения  $z$  являются арифметическими неизвестными. Для значений, отличных от арифметических неизвестных, могут потребоваться другие формы упрощения; например, если переменная  $z$  равна  $2 \times 3$ , то правильным упрощением выражения  $1 \times (0 + (2 \times 3))$  должно быть 6, а не  $2 \times 3$ .

Подводя итог, можно отметить, что основной процесс обучения на основе объяснения действует, как описано ниже.

- Используя рассматриваемый пример, составить доказательство того, что целиевой предикат является применимым к данному примеру, с использованием доступных фоновых знаний.
- Одновременно с этим сформировать обобщенное дерево доказательства для цели, описанной с помощью переменных, в котором используются такие же этапы логического вывода, как и в первоначальном дереве доказательства.

3. Сформулировать новое правило, левая часть которого состоит из листовых узлов дерева доказательства, а правая часть представляет собой цель, описанную с помощью переменных (после применения соответствующих связываний из обобщенного доказательства).
4. Удалить все условия, которые являются истинными независимо от значений переменных в цели.

### Повышение эффективности правила

Обобщенное дерево доказательства, показанное на рис. 19.5, фактически приводит к получению больше чем одного обобщенного правила. Например, если будет прекращено наращивание правой ветви в дереве доказательства после того, как одна ветвь достигнет этапа *Primitive*, т.е. если будет выполнено **отсечение** этой ветви, то получим следующее правило:

$$\text{Primitive}(z) \Rightarrow \text{Simplify}(1 \times (0 + z), z)$$

Это правило является таким же действительным, как и правило, в котором используется предикат *ArithmetcUnknown*, но более общим, поскольку оно охватывает также случаи, в которых  $z$  является числом. Из дерева доказательства можно извлечь еще более общее правило, выполняя отсечение ветви после этапа *Simplify*( $y + z, w$ ), что приводит к получению такого правила:

$$\text{Simplify}(y + z, w) \Rightarrow \text{Simplify}(1 \times (y + z), w)$$

Вообще говоря, из любого частичного поддерева обобщенного дерева доказательства можно извлечь какое-то правило. Это означает, что имеет место еще одна проблема: какое из этих правил следует выбрать?

Выбор правила, подлежащего выработке, сводится к решению вопроса об эффективности выбранного правила. Ниже описаны три фактора, которые касаются анализа прироста эффективности, достигнутого в результате обучения на основе объяснения.

1. Введение большого количества правил влечет за собой замедление процесса формирования рассуждений, поскольку механизм логического вывода так или иначе должен проверять эти правила, даже в тех случаях, когда они не приводят к решению. Иными словами, увеличение количества правил приводит к **увеличению коэффициента ветвления** в пространстве поиска.
2. Для того чтобы компенсировать замедление процесса формирования рассуждений, производные правила должны обеспечивать существенное увеличение скорости для тех случаев, которые они охватывают. Такое увеличение достигается в основном благодаря тому, что производные правила позволяют избегать тупиков, которые могли бы возникнуть без их использования, а также в связи с тем, что они сокращают само доказательство.
3. Производные правила должны быть как можно более общими, для того чтобы они применялись к максимально возможному множеству случаев.

Общепринятый подход к обеспечению эффективности таких производных правил состоит в том, чтобы следить за соблюдением требования, касающегося **операционной пригодности** каждой подцели в правиле. Подцель является операционно пригодной, если ее можно достаточно “легко” решить. Например, подцель

*Primitive(z)* решить легко, и для этого требуется самое большее два этапа вывода, а для достижения подцели *Simplify(y+z, w)* может потребоваться произвольный объем логического вывода в зависимости от значений *y* и *z*. Если проверка на операционную пригодность выполняется на каждом этапе формирования обобщенного доказательства, то появляется возможность отсечь оставшуюся ветвь, как только будет найдена операционно пригодная подцель, и оставить в качестве конъюнкта нового правила лишь эту операционно пригодную подцель.

К сожалению, обычно приходится искать компромисс между операционной пригодностью и общностью. Для более конкретных подцелей обычно бывает проще найти решение, но они охватывают меньше случаев. Кроме того, сам критерий операционной пригодности в определенной степени субъективен: можно уверенно сказать, что подцель, для достижения которой требуется один или два этапа, является операционно пригодной, но можно ли это утверждать, если количество этапов приближается к 10 или 100? Наконец, стоимость поиска решения для данной конкретной подцели зависит от того, какие еще правила имеются в базе знаний. По мере введения дополнительных правил эта стоимость может и увеличиваться, и уменьшаться. Таким образом, при эксплуатации систем EBL фактически приходится сталкиваться с очень сложными проблемами оптимизации, пытаясь добиться максимальной эффективности заданной начальной базы знаний. Иногда существует возможность вывести математическую модель влияния операции добавления какого-то конкретного правила на общую эффективность и использовать эту модель для определения наилучшего правила, подлежащего добавлению. Но такой анализ может стать очень сложным, особенно когда приходится сталкиваться с рекурсивными правилами. Один из перспективных подходов состоит в эмпирическом решении проблемы эффективности путем добавления нескольких правил и определения того, какое из них является полезным и действительно позволяет ускорить работу.

Эмпирический анализ эффективности фактически становится основой системы EBL. То, что в предыдущем абзаце неформально было названо “эффективностью заданной базы знаний”, в действительности представляет собой усредненную по рассматриваемым случаям сложность решения задач, предъявляемых системе и описываемых некоторым распределением вероятностей. Система EBL позволяет проводить обобщение по результатам решения примеров задач, встретившихся в прошлом, поэтому способствует повышению эффективности базы знаний применительно к тем типам задач, которые, скорее всего, должны встретиться в будущем. Такая организация работы способствует достижению успеха, при условии, что распределение вероятностей прошлых примеров приблизительно соответствует распределению вероятностей будущих примеров; это — такое же предположение, которое лежало в основе метода PAC-обучения, описанного в разделе 18.5. Если проведено тщательное и продуманное проектирование системы EBL, то появляется возможность достичь существенного ускорения работы. Например, очень крупная система обработки естественного языка на основе Prolog, предназначенная для прямого и обратного перевода речи со шведского на английский, позволила добиться приемлемой производительности работы в реальном времени только благодаря применению системы EBL в процессе синтаксического анализа [1351].

## 19.4. ОБУЧЕНИЕ С ИСПОЛЬЗОВАНИЕМ ИНФОРМАЦИИ О РЕЛЕВАНТНОСТИ

Путешественник, приехавший в Бразилию, о котором говорилось раньше, по-видимому, сумел прийти к достоверному обобщению, касающемуся языка, на котором говорят остальные бразильцы. Полученный этим путешественником логический вывод был основан на его фоновых знаниях, а именно на знаниях о том, что население данной конкретной страны (обычно) говорит на одном и тот же языке. Это утверждение можно выразить в логике первого порядка следующим образом<sup>3</sup>:

$$\text{Nationality}(x, n) \wedge \text{Nationality}(y, n) \wedge \text{Language}(x, l) \Rightarrow \text{Language}(y, l) \quad (19.6)$$

Это высказывание буквально означает: “Если  $x$  и  $y$  имеют одинаковое гражданство  $n$  и  $x$  говорит на языке  $l$ , то  $y$  также говорит на этом языке”. Нетрудно показать, что на основании этого высказывания и следующего наблюдения:

$$\text{Nationality}(\text{Fernando}, \text{Brazil}) \wedge \text{Language}(\text{Fernando}, \text{Portuguese})$$

можно прийти к такому заключению (см. упр. 19.1):

$$\text{Nationality}(x, \text{Brazil}) \Rightarrow \text{Language}(x, \text{Portuguese})$$

Высказывания, подобные приведенному в уравнении 19.6, выражают строгую форму релевантности: если известно гражданство человека, то полностью определен язык, на котором он говорит (эту мысль можно выразить иначе — язык определяется функциональной зависимостью от гражданства). Такие высказывания называются ~~функциональными зависимостями~~, или ~~определенными~~, и встречаются настолько часто в приложениях определенных типов (например, в спецификациях проектов баз данных), что для их записи используется специальный синтаксис. В этой главе мы будем придерживаться системы обозначений, предложенной Дэвисом [329]:

$$\text{Nationality}(x, n) \succ \text{Language}(x, l)$$

Как обычно, это обозначение представляет собой синтаксическое упрощение, но оно подчеркивает тот факт, что определение в действительности показывает связь между предикатами; в данном случае гражданство определяет язык. Аналогичным образом могут быть представлены релевантные свойства, определяющие проводимость и плотность:

$$\text{Material}(x, m) \wedge \text{Temperature}(x, t) \succ \text{Conductance}(x, p)$$

$$\text{Material}(x, m) \wedge \text{Temperature}(x, t) \succ \text{Density}(x, d)$$

Соответствующие обобщения следуют логически из определений и наблюдений.

### Определение пространства гипотез

Хотя определения позволяют обосновать общие заключения, касающиеся всех бразильцев или всех образцов меди, исследуемых при заданной температуре, они,

<sup>3</sup> Для упрощения в этой главе предполагается, что каждый человек говорит только на одном языке. Безусловно, в это правило потребовалось бы ввести поправку, если бы дело касалось таких многоязычных стран, как Швейцария или Индия.

безусловно, не позволяют создать на основании одного примера общую прогностическую теорию, которая распространялась бы на граждан всех государств или на любые материалы, исследуемые при любых температурах. По существу, основной эффект применения определений можно рассматривать как ограничение пространства гипотез, которые должен рассматривать обучающийся агент. В частности, согласно определению, при прогнозировании проводимости достаточно учитывать только материал и температуру, но игнорировать массу, права собственности на рассматриваемый образец, день недели, фамилию главы правительства и т.д. Но гипотезы, касающиеся проводимости, безусловно, могут включать термы, которые в свою очередь зависят от материала и температуры, в том числе учитывающие молекулярную структуру, тепловую энергию или плотность свободных электронов.  $\bowtie$  *Определения задают базовый словарь, достаточный для построения гипотез, касающихся целевого предиката.* Это утверждение можно доказать, продемонстрировав, что данное конкретное определение логически эквивалентно утверждению о том, что правильная формулировка целевого предиката является одной из множества всех формулировок, которые могут быть представлены с помощью предикатов из левой части определения.

Интуитивно ясно, что сокращение размера пространства гипотез должно способствовать упрощению задачи изучения целевого предиката. Возможный при этом прирост эффективности можно оценить с использованием основных результатов теории вычислительного обучения (см. раздел 18.5). Вначале напомним, что для булевых функций требуется  $\log(|\mathbf{H}|)$  примеров, чтобы свести все пространство гипотез к одной приемлемой гипотезе, где  $|\mathbf{H}|$  — размер пространства гипотез. Если в распоряжении ученика имеется  $n$  булевых характеристик, для которых должны быть построены гипотезы, то в отсутствии дополнительных ограничений  $|\mathbf{H}| = O(2^{2^n})$ , поэтому необходимое количество примеров составляет  $O(2^n)$ . Если же в левой части определения находятся  $d$  предикатов, то для ученика потребуется только  $O(2^d)$  примеров, что соответствует сокращению в  $O(2^{n-d})$  раз. Для смещенного пространства гипотез, такого как конъюнктивно смещенное пространство, это сокращение будет не столь радикальным, но все еще достаточно существенным.

### Обучение и использование информации о релевантности

Как было указано во введении к данной главе, априорные знания являются полезными при обучении, но они также должны быть освоены в процессе обучения. Поэтому для обеспечения целостного подхода к обучению с учетом релевантности необходимо предусмотреть алгоритм обучения, относящийся к определениям. Алгоритм обучения, приведенный в этом разделе, представляет собой прямолинейную попытку найти простейшее определение, совместимое с рассматриваемыми результатами наблюдений. В частности, определение  $P \succ Q$  говорит о том, что если какие-либо примеры согласуются по выражению  $P$ , то они должны также согласовываться по выражению  $Q$ . Поэтому определение является совместимым с множеством примеров, если каждая пара, которая согласуется по предикатам в левой части определения, согласуется также по целевому предикату, т.е. входит в одну и ту же классификацию. Например, предположим, что имеются примеры результатов измерения проводимости образцов материалов, приведенные в табл. 19.1.

Минимальным согласованным определением является *Material \ Temperature \ Conductance*. Существует также одно не минимальное, но совместимое определе-

ние, а именно *Mass\Size\Temperature-Conductance*. Оно совместимо с примерами, поскольку масса и размер определяют плотность, а в рассматриваемом наборе данных отсутствуют два разных материала с одной и той же плотностью. Как обычно, для устранения гипотез, которые только внешне кажутся правильными, необходимо иметь более крупное множество примеров.

**Таблица 19.1. Примеры результатов измерения проводимости образцов материалов**

Образец	Масса	Температура	Материал	Размер	Проводимость
S1	12	26	Медь	3	0,59
S1	12	100	Медь	3	0,57
S2	24	26	Медь	6	0,59
S3	12	26	Свинец	2	0,05
S3	12	100	Свинец	2	0,04
S4	24	26	Свинец	4	0,05

Для поиска минимальных совместимых определений можно применить несколько подходящих для этого алгоритмов. Наиболее очевидный подход состоит в проведении поиска в пространстве определений и проверке всех определений с одним предикатом, двумя предикатами и т.д. до тех пор, пока не будет найдено совместимое определение. Предполагается, что используется простое представление на основе атрибутов, аналогичное тому, которое использовалось при обучении деревьев решений в главе 18. Определение *d* будет представлено с помощью множества атрибутов в левой части, поскольку предполагается, что целевой предикат остается постоянным. Основной алгоритм, соответствующий этому подходу, приведен в листинге 19.3.

#### Листинг 19.3. Алгоритм поиска минимального совместимого определения

```

function Minimal-Consistent-Det(E, A) returns множество атрибутов
  inputs: E, множество примеров
           A, множество атрибутов с количеством элементов n

  for i  $\leftarrow$  0, ..., n do
    for each подмножество Ai множества A
      с количеством элементов i do
        if Consistent-Det?(Ai, E) then return Ai

function Consistent-Det?(A, E) returns истинностное значение
  inputs: A, множество атрибутов
           E, множество примеров
  local variables: H, хэш-таблица

  for each пример e in E do
    if некоторый пример в таблице H имеет такое же значение
      множества атрибутов A, что и пример e, но имеет другую
      классификацию then return false
    сохранить обозначение класса примера e в таблице H под
      индексом, соответствующим значениям атрибутов A примера e
  return true

```

Временная сложность этого алгоритма зависит от размера наименьшего совместного определения. Предположим, что это определение включает  $p$  атрибутов из общего количества, равного  $n$  атрибутам. В таком случае алгоритм не отыщет данное определение до тех пор, пока не выполнит поиск во всех подмножествах  $A$  с размером  $p$ . Количество таких подмножеств измеряется значением

$${n \choose p} = O(n^p),$$

поэтому сложность алгоритма зависит экспоненциально от размера минимального определения. Как оказалось, данная задача является NP-полной, поэтому не следует рассчитывать на то, что в общем случае будет достигнута более высокая производительность. Тем не менее во многих проблемных областях существует локальная структура (понятие локально структурированных проблемных областей рассматривается в главе 14), благодаря которой  $p$  становится небольшим.

Получив в свое распоряжение алгоритм изучения определений, обучающийся агент приобретает способность создавать минимальную гипотезу, с помощью которой он может изучать целевой предикат. Например, можно скомбинировать алгоритм Minimal-Consistent-Det с алгоритмом Decision-Tree-Learning. Это приведет к созданию алгоритма обучения дерева решений с учетом релевантности, RBDTL (Relevance-Based Decision-Tree Learning), в котором вначале определяется минимальное множество релевантных атрибутов, а затем это множество передается алгоритму формирования дерева решений для обучения. В отличие от алгоритма Decision-Tree-Learning, алгоритм RBDTL одновременно обеспечивает и обучение, и использование информации о релевантности для минимизации своего пространства гипотез. Можно рассчитывать на то, что алгоритм RBDTL даст возможность проводить обучение быстрее по сравнению с алгоритмом Decision-Tree-Learning, и действительно так и происходит. На рис. 19.6 показана производительность обучения для этих двух алгоритмов на основе случайно сгенерированных данных для функции, которая зависит только от 5 из 16 атрибутов. Ясно, что в тех случаях, когда все доступные атрибуты являются релевантными, алгоритм RBDTL не показывает каких-либо преимуществ.

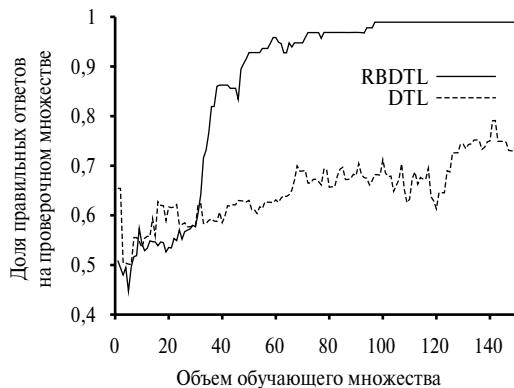


Рис. 19.6. Сравнение производительности алгоритмов RBDTL и Decision-Tree-Learning при обработке случайно сгенерированных данных применительно к целевой функции, которая зависит только от 5 из 16 атрибутов

В данном разделе приведены лишь самые основные сведения из области исследования **декларативного смещения**, задача которой состоит в изучении того, как могут использоваться априорные знания для выявления приемлемого пространства гипотез, в котором должен осуществляться поиск правильного целевого определения. В нем не даны ответы на многие вопросы, например, на те, которые приведены ниже.

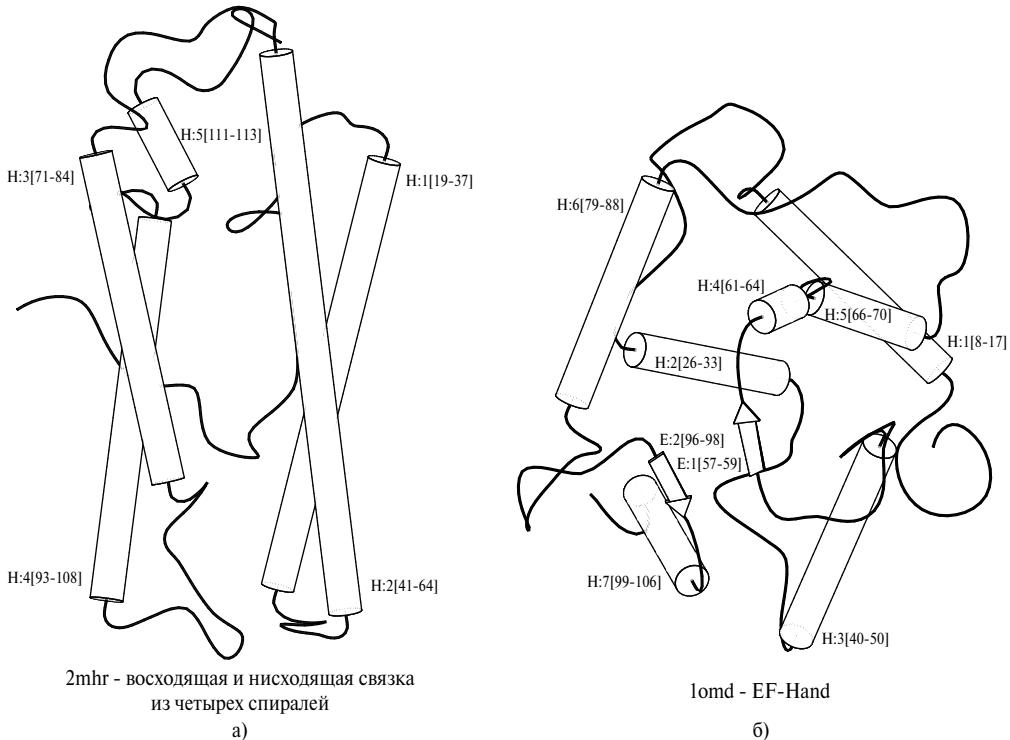
- Как могут быть дополнены эти алгоритмы для того, чтобы в них учитывался шум?
- Можно ли обеспечить обработку переменных с непрерывными значениями?
- Можно ли использовать другие виды априорных знаний, кроме определений?
- Как можно обобщить эти алгоритмы для охвата любой теории первого порядка, а не только представлений на основе атрибутов?

Некоторые из этих вопросов рассматриваются в следующем разделе.

## 19.5. ИНДУКТИВНОЕ ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

В индуктивном логическом программировании (Inductive Logic Programming — ILP) объединяются индуктивные методы с мощными представлениями в логике первого порядка, и оно в основном сосредоточивается на представлении теорий в виде логических программ<sup>4</sup>. Это научное направление получило широкую известность по трем причинам. Во-первых, индуктивное логическое программирование лежит в основе строгого подхода к решению общих задач индуктивного обучения на основе базы знаний. Во-вторых, в рамках этого направления созданы полные алгоритмы логического вывода общих теорий в логике первого порядка с использованием примеров; это означает, что данные алгоритмы могут успешно применяться для обучения в тех проблемных областях, в которых использование алгоритмов на основе атрибутов связано с определенными затруднениями. В качестве соответствующего примера можно привести применение методов ILP в изучении того, как сворачиваются белковые структуры (рис. 19.7). Трехмерную конфигурацию молекулы белка невозможно представить с помощью какого-либо приемлемого способа в виде множества атрибутов, поскольку конфигурация молекулы неявно обусловлена связями между объектами, а не атрибутами отдельного объекта. Одним из языков, подходящих для описания таких связей, является логика первого порядка. В-третьих, с помощью методов индуктивного логического программирования могутрабатываться гипотезы, которые являются (относительно) простыми для восприятия людьми. Например, перевод на естественный язык правила, приведенного в подпункте к рис. 19.7, может уточняться и подвергаться критике биологами, которые работают в данной области. Это означает, что системы индуктивного логического программирования могут участвовать в научном цикле экспериментирования, выработки гипотез, обсуждения и опровержения. Такое участие невозможно при использовании таких систем, которые вырабатывают классификаторы в виде “черных ящиков”, например нейронных сетей.

<sup>4</sup> На данном этапе читателю может потребоваться обратиться к главе 9 для повторного ознакомления с некоторыми из основных понятий, включая хорновские выражения, конъюнктивную нормальную форму, унификацию и резолюцию.



*Рис. 19.7. Пример решения задачи обучения: положительный и отрицательный примеры применения понятия “восходящей и нисходящей связки из четырех спиралей” в проблемной области сворачивания белка (а), (б). Пример каждой структуры кодируется в виде логического выражения, состоящего примерно из 100 конъюнктоов, такого как  $\text{TotalLength}(\text{D2mhr}, 118) \wedge \text{NumberHelices}(\text{D2mhr}, 6) \wedge \dots$ . На основании этих описаний и таких классификаций, как Fold(Four-Helical-Up-And-Down-Bundle, D2mhr), система индуктивного логического программирования Progol [1098] сформировала по методу обучения следующее правило:*

```
Fold(Four-Helical-Up-And-Down-Bundle, p) ⇐
    Helix(p, h1) ∧ Length(h1, High) ∧ Position(p, h1, n)
    ∧ (1 ≤ n ≤ 3) ∧ Adjacent(p, h1, h2) ∧ Helix(p, h2)
```

*Правило такого рода невозможно выработать путем обучения или даже представить с помощью любых механизмов, основанных на атрибутах, наподобие тех, которые рассматривались в предыдущих главах. Это правило можно перевести на естественный язык, как показано ниже.*

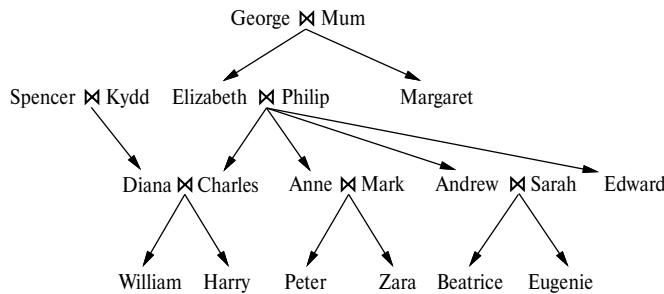
*Белок P относится к классу методов сворачивания “связка из четырех восходящих и нисходящих спиралей”, если он включает длинную спираль h<sub>1</sub> во второй структурной позиции между позициями 1 и 3 и если спираль h<sub>1</sub> находится рядом со второй спиралью*

### Практический пример

Как было указано выше, из уравнения 19.5 следует, что общая задача индукции на основе знаний состоит в “решении” ограничения логического следствия для неизвестной гипотезы *Hypothesis*, если даны фоновые знания *Background* и примеры, представленные с помощью описаний *Descriptions* и классификаций *Classifications*:

*Background*  $\wedge$  *Hypothesis*  $\wedge$  *Descriptions*  $\models$  *Classifications*

Для иллюстрации этого процесса рассмотрим задачу изучения семейных связей на примерах. Описания будут состоять из развернутого генеалогического дерева, описанного в терминах отношений *Mother* (Мать), *Father* (Отец) и *Married* (Состоит в браке), а также свойств *Male* (Мужчина) и *Female* (Женщина). В качестве примера будет использоваться генеалогическое дерево, рассматриваемое в упр. 8.11, которое показано на рис. 19.8. Соответствующие описания приведены ниже.



*Father(Philip, Charles)*

*Father(Philip, Anne)*

...

*Mother(Mum, Margaret)*

*Mother(Mum, Elizabeth)*

...

*Married(Diana, Charles)*

*Married(Elizabeth, Philip)*

...

*Male(Philip)*

*Male(Charles)*

...

*Female(Beatrice)*

*Female(Margaret)*

...

Высказывания, входящие в состав классификаций *Classifications*, зависят от изучаемого целевого понятия. Например, может потребоваться изучить понятия *Grandparent* (Дедушка или бабушка), *BrotherInLaw* (Двоюродный брат) или *Ancestor* (Предок). Что касается понятия *Grandparent*, то полное множество *Classifications* содержит  $20 \times 20 = 400$  конъюнктов в следующей форме:

*Grandparent(Mum, Charles)*

*Grandparent(Elizabeth, Beatrice)*

...

$\neg$ *Grandparent(Mum, Harry)*

$\neg$ *Grandparent(Spencer, Peter)*

...

Безусловно, обучение может быть проведено на основе какого-то подмножества этого полного множества.

Задачей применения любой программы индуктивного обучения является выработка множества высказываний, соответствующих гипотезе *Hypothesis*, такого, что в нем выполняется заданное ограничение логического следствия. На данный момент предположим, что у агента отсутствуют фоновые знания: отношение *Background* является пустым. В таком случае одно из возможных решений для *Hypothesis* состоит в следующем:

$$\begin{aligned} \text{Grandparent}(x, y) \Leftrightarrow & [\exists z \text{ Mother}(x, z) \wedge \text{Mother}(z, y)] \\ \vee & [\exists z \text{ Mother}(x, z) \wedge \text{Father}(z, y)] \\ \vee & [\exists z \text{ Father}(x, z) \wedge \text{Mother}(z, y)] \\ \vee & [\exists z \text{ Father}(x, z) \wedge \text{Father}(z, y)] \end{aligned}$$

Следует отметить, что алгоритм обучения на основе атрибутов, такой как Decision-Tree-Learning, в попытке решить эту задачу бесконечно углубится в дерево, но не достигнет успеха. Для того чтобы выразить отношение *Grandparent* в виде атрибута (т.е. унарного предиката), необходимо будет преобразовывать пары людей в объекты:

$$\text{Grandparent}(\langle \text{Mum}, \text{Charles} \rangle) \dots$$

Затем мы окажемся в тупике, пытаясь представить описания примеров. Единственно возможными атрибутами станут такие ужасно неуклюжие конструкции, как следующая:

$$\text{FirstElementIsMotherOfElizabeth}(\langle \text{Mum}, \text{Charles} \rangle)$$

Определением отношения *Grandparent* в терминах этих атрибутов становится большая дизъюнкция описаний конкретных случаев, которая вообще не может быть обобщена с учетом новых примеров.  $\bowtie$  Алгоритмы обучения на основе атрибутов не способны обеспечить изучение реляционных предикатов. Таким образом, одним из принципиальных преимуществ алгоритмов ILP является их применимость для решения гораздо более широкого диапазона задач, включая реляционные задачи.

Читатель должен был, безусловно, заметить, что для представления определения *Grandparent* мог бы помочь небольшой фрагмент фоновых знаний. Например, если в состав фоновых знаний *Background* входит следующее высказывание:

$$\text{Parent}(x, y) \Leftrightarrow [\text{Mother}(x, y) \vee \text{Father}(x, y)]$$

то определение отношения *Grandparent* можно свести к следующему:

$$\text{Grandparent}(x, y) \Leftrightarrow [\exists z \text{ Parent}(x, z) \wedge \text{Parent}(z, y)]$$

Этот пример показывает, насколько существенно фоновые знания позволяют сократить размер гипотез, требуемых для объяснения результатов наблюдений.

Алгоритмы ILP позволяют также создавать новые предикаты для упрощения способов выражения объяснительных гипотез. Применительно к примеру данных, приведенному выше, было бы вполне резонно, чтобы программа ILP предложила дополнительный предикат, который можно было бы назвать “*Parent*” (Родитель), для упрощения определений целевых предикатов. Алгоритмы, которые способны вырабатывать новые предикаты, называются алгоритмами  $\bowtie$  конструктивной индукции. Очевидно, что конструктивная индукция является необходимой частью подхода к обучению на основе накопления знаний, который был кратко описан во введении. Задача накопления знаний всегда была одной из сложнейших задач машинного обучения, но некоторые методы ILP предоставляют эффективные механизмы ее решения.

В оставшейся части данной главы исследуются два принципиальных подхода к индуктивному логическому программированию. В первом из них используется обобщение методов на основе деревьев решений, а во втором используются методы, основанные на инвертировании доказательства с помощью резолюции.

### Нисходящие методы индуктивного обучения

Описанный здесь первый подход к индуктивному логическому программированию начинается с получения очень общего правила и его постепенного уточнения для достижения соответствия с данными. По сути такой же процесс происходит при обучении деревьев решений, когда дерево решений постепенно наращивается до тех пор, пока не становится совместимым с результатами наблюдений. Но для осуществления индуктивного логического программирования вместо атрибутов используются литералы первого порядка, а вместо дерева решений рассматриваются гипотезы, представляющие собой множества выражений. В этом разделе рассматривается программа *Foil* [1258], одна из первых программ ILP.

Предположим, что предпринимается попытка изучить определение предиката *Grandfather*(*x, y*) с использованием тех же данных о семейных отношениях, как и перед этим. По такому же принципу, как и при обучении деревьев решений, эти примеры можно разделить на положительные и отрицательные. Положительными примерами являются следующие:

*⟨George, Anne⟩, ⟨Philip, Peter⟩, ⟨Spencer, Harry⟩, ...*

а отрицательными — такие примеры:

*⟨George, Elizabeth⟩, ⟨Harry, Zara⟩, ⟨Charles, Philip⟩, ...*

Обратите внимание на то, что каждый пример представляет собой пару объектов, поскольку *Grandfather* — это бинарный предикат. В целом в данном генеалогическом дереве имеется 12 положительных примеров и 388 отрицательных (все прочие пары людей).

Программа *Foil* строит множества выражений, в каждом из которых *Grandfather*(*x, y*) является головой предиката. Эти выражения должны классифицировать 12 положительных примеров как экземпляры отношения *Grandfather*(*x, y*) и вместе с тем исключать 388 отрицательных примеров. Рассматриваемые выражения являются хорновскими выражениями, расширенными с помощью отрицаемых литералов, в которых используется отрицание как недостижение цели, по аналогии с языком Prolog. Первоначальное выражение имеет такое пустое тело:

$\Rightarrow Grandfather(x, y)$

Это выражение классифицирует каждый пример как положительный, поэтому требует уточнения. Такую операцию можно выполнить, каждый раз вводя по одному литералу в левую часть выражения. Ниже перечислены три возможных дополнения.

*Father(x, y)  $\Rightarrow Grandfather(x, y)$*

*Parent(x, z)  $\Rightarrow Grandfather(x, y)$*

*Father(x, z)  $\Rightarrow Grandfather(x, y)$*

(Обратите внимание на то, что здесь предполагается, будто выражение, определяющее предикат *Parent*, уже является частью фоновых знаний.) Первое из этих

трех выражений неправильно классифицирует все 12 положительных примеров как отрицательные и поэтому может быть проигнорировано. Второе и третье выражения согласуются со всеми положительными примерами, но второе является неправильным применительно к большей части отрицательных примеров (этих примеров вдвое больше, чем в первом случае, поскольку в них допускается произвольная подстановка данных и о материях, и об отцах). Поэтому предпочтительным является третье выражение.

Теперь необходимо дополнительно уточнить это выражение, чтобы исключить те случаи, в которых  $x$  является отцом некоторого  $z$ , но  $z$  не является родителем  $y$ . Добавление единственного литерала  $\text{Parent}(z, y)$  позволяет получить следующее выражение, которое правильно классифицирует все примеры:

$$\text{Father}(x, z) \wedge \text{Parent}(z, y) \Rightarrow \text{Grandfather}(x, y)$$

Программа Foil способна найти и выбрать этот литерал, решая тем самым задачу обучения. Вообще говоря, программе Foil обычно приходится выполнять поиск среди многих неподходящих выражений, прежде чем она сможет найти правильное решение.

Этот пример может служить очень простой иллюстрацией того, как действует программа Foil. Набросок общего алгоритма этой программы приведен в листинге 19.4. По существу, алгоритм повторно составляет выражение, вводя в него один литерал за другим до тех пор, пока это выражение не начинает соглашаться с некоторым подмножеством положительных примеров и при этом не согласуется ни с одним отрицательным примером. Затем положительные примеры, охваченные данным выражением, удаляются из обучающего множества и процесс продолжается до тех пор, пока не остается ни одного положительного примера. Двумя основными процедурами, требующими пояснения, являются New-Literal, которая создает все возможные новые литералы для добавления к выражению, и Choose-Literal, которая выбирает литерал для добавления.

**Листинг 19.4. Набросок алгоритма Foil, применяемого для изучения множеств хорновских выражений первого порядка на основе примеров. Функции New-Literal и Choose-Literal описаны в тексте**

---

```

function Foil(examples, target) returns множество хорновских выражений
  inputs: examples, множество примеров
          target, литерал для целевого предиката
  local variables: clauses, множество выражений, первоначально пустое

  while множество examples содержит положительные примеры do
    clause ← New-Clause(examples, target)
    удалить примеры, охваченные выражением clause,
    из множества examples
    добавить выражение clause к множеству clauses
  return clauses

function New-Clause(examples, target) returns хорновское выражение
  local variables: clause, выражение, головой которого является
                  target, а тело пусто
                  l, литерал, который должен быть добавлен
                  к выражению clause
                  extended_examples, множество примеров со
                  значениями для новых переменных

```

```

extended_examples ← examples
while extended_examples содержит отрицательные примеры do
     $l \leftarrow \text{Choose-Literal}(\text{New-Literals}(clause), extended\_examples)$ 
    добавить  $l$  к телу выражения  $clause$ 
    extended_examples ← множество примеров, созданных путем
        применения функции Extend-Example
        к каждому примеру в множестве
        extended_examples

return  $clause$ 

function Extend-Example(example, literal) returns множество примеров
    if пример example согласуется с литералом literal
        then return множество примеров, созданное путем дополнения
            примера example каждым возможным значением
            константы для каждой новой переменной в литерале
            literal
    else return пустое множество

```

---

Процедура New-Literals выбирает некоторое выражение и составляет все возможные “полезные” литералы, которые могут быть добавлены к этому выражению. Рассмотрим в качестве примера следующее выражение:

$$\text{Father}(x, z) \Rightarrow \text{Grandfather}(x, y)$$

К нему могут быть добавлены три описанные ниже разновидности литералов.

- Литералы, в которых используются предикаты.** Литерал может быть отрицаемым или неотрицаемым, в нем может применяться любой существующий предикат (включая целевой предикат), а все параметры должны быть переменными. В качестве любого параметра предиката может использоваться любая переменная, с учетом одного ограничения: каждый литерал должен включать по меньшей мере одну переменную из предыдущего литерала или из головы выражения. Такие литералы, как  $\text{Mother}(z, u)$ ,  $\text{Married}(z, y)$ ,  $\neg\text{Male}(y)$  и  $\text{Grandfather}(v, x)$ , являются допустимыми, а  $\text{Married}(u, v)$  — нет. Обратите внимание на то, что благодаря имеющейся возможности применять предикат из головы выражения программа Foil может использоваться для изучения рекурсивных определений.
- Литералы равенства и неравенства.** Эти литералы связывают переменные, уже присутствующие в данном выражении. Например, можно ввести литерал  $z \neq x$ . Данные литералы могут также включать константы, заданные пользователем. Для изучения арифметических выражений могут использоваться константы 0 и 1, а для изучения функций, заданных на списках, — пустой список [ ].
- Арифметические сравнения.** При обработке функций непрерывных переменных могут быть добавлены такие литералы, как  $x > y$  и  $y \leq z$ . По аналогии с обучением деревьев решений могут быть выбраны пороговые значения для максимизации различительной мощи проверки.

Возникающий в результате коэффициент ветвления в этом пространстве поиска очень велик (см. упр. 19.6), но в программе Foil для его уменьшения может также использоваться информация о типе. Например, если проблемная область включает

данные не только о числах, но и о людях, то ограничения типов позволяют исключить возможность выработки в процедуре `New-Literals` таких литералов, как `Parent(x, n)`, где  $x$  — человек, а  $n$  — число.

В процедуре `Choose-Literal` используется эвристика, немного напоминающая эвристику, основанную на приросте информации (см. с. 878), для определения того, какой литерал должен быть выбран для добавления. Точные сведения о том, как это осуществляется, не имеет в контексте данной главы слишком большого значения, к тому же было опробовано много различных вариантов таких эвристик. Одно интересное дополнительное средство программы `Foil` состоит в использовании принципа бритвы Оккама для устраниния некоторых гипотез. Если некоторое выражение становится длиннее (согласно определенной метрике), чем общая длина положительных примеров, объясняемых с помощью этого выражения, оно больше не рассматривается в качестве потенциальной гипотезы. Такой метод предоставляет удобный способ предотвращения создания чрезмерно сложных выражений, которые обусловлены наличием шума в данных. Объяснение связи между шумом и длиной выражений приведено на с. 949.

Программа `Foil` и аналогичные ей программы использовались для изучения широкого ряда определений. Одна из наиболее впечатляющих демонстраций возможностей таких программ [1260] касалась решения длинной последовательности упражнений по функциям обработки списков из учебника по языку Prolog, написанного Братко [174]. В каждом случае программа показала свою способность найти в процессе обучения правильное определение функции на основе небольшого множества примеров, используя в качестве фоновых знаний ранее изученные функции.

### Индуктивное обучение с помощью обратной дедукции

Второй важный подход к индуктивному логическому программированию предусматривает обращение обычного процесса дедуктивного доказательства. Метод **обратной резолюции** основан на том наблюдении, что если классификация *Classifications* примера следует из выражения *Background* $\wedge$ *Hypothesis* $\wedge$ *Descriptions*, то должна существовать возможность доказать этот факт с помощью метода резолюции (поскольку метод резолюции является полным). А если есть возможность “проводить доказательство в обратном направлении”, то можно найти такую гипотезу *Hypothesis*, через которую проходит это доказательство. Поэтому задача состоит в том, чтобы найти способ обращения процесса резолюции.

В этом разделе будет показан процесс обратного доказательства для инверсного правила резолюции, который состоит из отдельных обратных этапов. Напомним, что в обычном этапе резолюции берутся два выражения,  $C_1$  и  $C_2$ , и к ним применяется операция уничтожения взаимно противоположных литералов (операция резолюции) для получения **результаты**  $C$ . С другой стороны, на этапе инверсной резолюции берется результат  $C$  и формируются два выражения,  $C_1$  и  $C_2$ , такие, что  $C$  является результатом применения операции уничтожения взаимно противоположных литералов в выражениях  $C_1$  и  $C_2$ . Еще один вариант состоит в том, что можно взять результат  $C$  и выражение  $C_1$  и сформировать выражение  $C_2$ , такое, что  $C$  является результатом применения операции резолюции к  $C_1$  и  $C_2$ .

Начальные этапы процесса обратной резолюции показаны на рис. 19.9; на этих этапах вся работа сосредоточивается на положительном примере *Grandparent*

(*George, Anne*). Процесс начинается с конца доказательства (который обозначен квадратом в нижней части рисунка). Предполагается, что резольвента  $C$  представляет собой пустое выражение (т.е. противоречие), а в качестве  $C_2$  берется выражение  $\neg Grandparent(George, Anne)$ , которое является отрицанием целевого примера. В первом обратном этапе используются выражения  $C$  и  $C_2$  и формируется выражение  $Grandparent(George, Anne)$ , соответствующее  $C_1$ . На следующем этапе это выражение берется в качестве  $C$ , выражение  $Parent(Elizabeth, Anne)$  используется в качестве  $C_2$ , а в качестве  $C_1$  формируется следующее выражение:

$$\neg Parent(Elizabeth, y) \vee Grandparent(George, y)$$

На последнем этапе данное выражение рассматривается как резольвента. Если в качестве  $C_2$  берется выражение  $Parent(George, Elizabeth)$ , то одним из возможных выражений  $C_1$  становится гипотеза:

$$Parent(x, z) \wedge Parent(z, y) \Rightarrow Parent(x, y)$$

Теперь в нашем распоряжении имеется доказательство с помощью резолюции, что из данной гипотезы, описаний и фоновых знаний следует классификация  $Grandparent(George, Anne)$ .

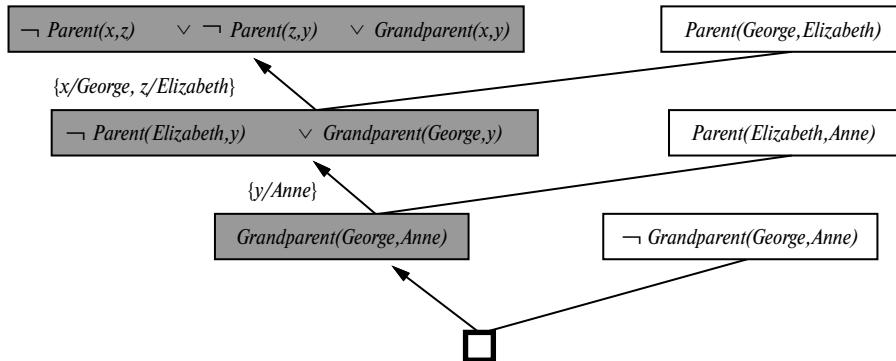


Рис. 19.9. Начальные этапы в процессе обратной резолюции. Затененные выражения сформированы на этапах обратной резолюции из выражений, находящихся справа и внизу. Незатененные выражения взяты из множеств Descriptions и Classifications

Очевидно, что в процессе обратной резолюции требуется поиск. Каждый этап обратной резолюции является недетерминированным, поскольку для любого выражения  $C$  может существовать большое или даже бесконечное количество выражений  $C_1$  и  $C_2$ , которые в результате операции резолюции преобразуются в  $C$ . Например, вместо применения выражения  $\neg Parent(Elizabeth, y) \vee Grandparent(George, y)$  в качестве  $C_1$  на последнем этапе, показанном на рис. 19.9, на этом этапе обратной резолюции можно было выбрать любое из следующих высказываний (см. упр. 19.4 и 19.5):

- $\neg Parent(Elizabeth, Anne) \vee Grandparent(George, Anne)$
- $\neg Parent(z, Anne) \vee Grandparent(George, Anne)$
- $\neg Parent(z, y) \vee Grandparent(George, y)$
- ...

Более того, выражения, которые участвуют на каждом этапе, могут быть выбраны из фоновых знаний *Background*, из описаний примеров *Descriptions*, из отрицаемых выражений в множестве *Classifications* или из выражений, выдвинутых в качестве гипотезы, которые уже были сформированы в дереве обратной резолюции. Такое большое количество возможностей означает, что без дополнительного управления возникает большой коэффициент ветвления (и поэтому поиск становится неэффективным). В реализованных на практике системах ILP был опробован целый ряд подходов к управлению поиском, в том числе подходы, описанные ниже.

1. Могут быть удалены избыточные варианты выбора, например, путем формирования только наиболее конкретных гипотез из всех возможных и соблюдения требования о том, чтобы все выражения, принятые в качестве гипотез, были совместимы и друг с другом, и с результатами наблюдений. Применение последнего критерия позволило бы исключить выражение  $\neg Parent(z, y) \vee Grandparent(George, y)$ , приведенное выше.
2. Могут быть наложены ограничения на стратегию доказательства. Например, в главе 9 было показано, что **линейная резолюция** — это полная, ограниченная стратегия, которая позволяет создавать только такие деревья доказательства, которые имеют линейную структуру ветвления (как на рис. 19.9).
3. Могут быть наложены ограничения на язык представления, например, путем устранения функциональных символов или применения только хорновских выражений. Например, язык Progol оперирует с хорновскими выражениями с использованием **обратного логического следствия**. Идея этого метода состоит в том, что ограничение логического следствия:

$Background \wedge Hypothesis \wedge Descriptions \models Classifications$

должно быть преобразовано в такую логически эквивалентную форму:

$Background \wedge Descriptions \wedge \neg Classifications \models \neg Hypothesis$

Исходя из этой формы представления, для вывода гипотезы *Hypothesis* можно использовать процесс, аналогичный обычной дедукции Prolog с помощью хорновских выражений, в котором применяется отрицание как недостижение цели. Поскольку данный метод ограничивается хорновскими выражениями, он является неполным, но может оказаться более эффективным, чем полная резолюция. Кроме того, при обратном логическом следствии появляется возможность использовать полный логический вывод [717].

4. Логический вывод может осуществляться с помощью проверки по модели, а не с помощью доказательства теоремы. В системе Progol [1098] для ограничения поиска используется определенная форма проверки по модели. Это означает, что в этой системе, как и в программировании множества ответов,рабатываются все возможные значения логических переменных, которые затем проверяются на совместимость.
5. Логический вывод может осуществляться с помощью базовых пропозициональных выражений, а не выражений в логике первого порядка. Система Linus [897] действует по принципу преобразования теорий логики первого порядка в выражения пропозициональной логики, поиска для этих выражений решений с помощью пропозициональной обучающейся системы, а затем

обратного преобразования. Как было показано на примере алгоритма SATplan в главе 11, при решении некоторых задач может оказаться более эффективной организация работы с помощью пропозициональных формул.

### Совершение открытий с помощью индуктивного логического программирования

Любая процедура обратной резолюции, в которой стратегия полной резолюции действует в противоположном направлении, в принципе представляет собой полный алгоритм изучения теорий первого порядка. Это означает, что если на основе некоторой неизвестной гипотезы *Hypothesis* вырабатывается множество примеров, то процедура обратной резолюции может выработать гипотезу *Hypothesis* из этих примеров. Такое замечание подсказывает любопытную возможность: предположим, что имеющиеся примеры включают данные о разнообразных траекториях падающих тел. Существует ли такая теоретическая возможность, что программа обратной резолюции позволит вывести логическим путем закон гравитации? Очевидно, что ответ на этот вопрос является положительным, поскольку именно закон гравитации позволяет объяснять все примеры траекторий при использовании подходящего вспомогательного математического аппарата. Аналогичным образом, вполне можно представить себе, что в пределах возможностей программ ILP находится открытие законов электромагнитного поля, квантовой механики и теории относительности. Разумеется, открытие этих законов находится и в пределах возможностей такой экспериментальной установки, в которой обезьяну посадили за пишущую машинку; нужны лишь более качественные эвристики и новые способы структуризации пространства поиска.

Но не углубляясь в эти крайности, можно отметить, что системы обратной резолюции действительно позволяют совершать небольшие, но важные открытия — создавать новые предикаты. Такую их способность часто рассматривают как нечто магическое, поскольку компьютеры принято считать устройствами, “просто работающими с тем, что им задано”. Но в действительности новые предикаты непосредственно следуют из этапов обратной резолюции. Простейшим случаем создания таких предикатов является преобразование в гипотезы двух новых выражений  $C_1$  и  $C_2$  на основе выражения  $C$ . Применение операции резолюции к выражениям  $C_1$  и  $C_2$  приводит к удалению в этих двух выражениях общего литерала, поэтому вполне возможно, что удаленный литерал содержал предикат, который не присутствовал в  $C$ . Таким образом, при проведении доказательства в обратном направлении одна из возможностей состоит в формировании нового предиката, из которого реконструируется недостающий литерал.

На рис. 19.10 показан пример, в котором новый предикат  $P$  формируется в процессе изучения определения для отношения *Ancestor*. После его формирования предикат  $P$  может использоваться на дальнейших этапах обратной резолюции. Например, на одном из дальнейших этапов может быть выдвинута гипотеза, что  $Mother(x, y) \Rightarrow P(x, y)$ . Таким образом, новый предикат  $P$  имеет смысл, определяемый процессом выработки гипотез, в которых он участвует. А обработка еще одного примера может привести к созданию ограничения  $Father(x, y) \Rightarrow P(x, y)$ . Иными словами, предикат  $P$  представляет собой описание того, что обычно рассматривается как отношение *Parent*. Как было указано выше, введение новых пре-

дикатов позволяет существенно уменьшить размер определения целевого предиката. Это означает, что системы обратной резолюции часто позволяют решать задачи обучения, неразрешимые с помощью других методов, поскольку предоставляют возможность изобретать новые предикаты.

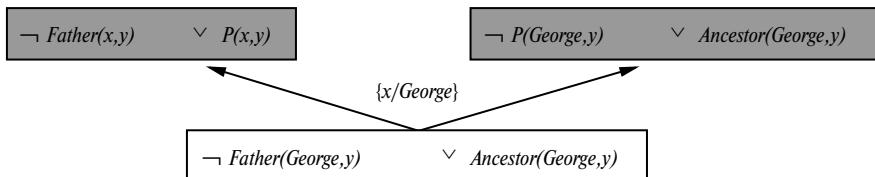


Рис. 19.10. Этап обратной резолюции, на котором вырабатывается новый предикат  $P$

Некоторые из самых радикальных революций в науке стали следствием изобретения новых предикатов и функций; в качестве примера можно привести открытие Галилеем принципа ускорения или уточнение Джоулем понятия тепловой энергии. Как только в распоряжении ученых оказываются эти новые определения, открытие новых законов становится относительно простым. При этом труднее всего понять, что какая-то новая сущность, имеющая конкретную связь с уже известными сущностями, позволяет объяснить целый класс результатов наблюдений с помощью гораздо более простой и изящной теории по сравнению с теми теориями, которые применялись раньше.

До сих пор с помощью систем ILP еще не были сделаны открытия на уровне Галилея или Джоуля, но даже небольшие открытия, сделанные с их помощью, оказались достойными публикации в научной литературе. Например, в *Journal of Molecular Biology* были описаны результаты автоматизированного открытия правил сворачивания белка с помощью программы Progol, действующей по принципу индуктивного логического программирования [1517]. Многие правила, открытые программой Progol, и раньше можно было вывести из известных принципов, но большинство из них еще не было до сих пор опубликовано в составе какой-либо стандартной биологической базы данных (один из примеров такого правила приведен на рис. 19.7). В одной из работ, относящихся к близкой теме, рассматривалась проблема открытия правил определения мутагенной способности нитроароматических веществ, обусловленной их молекулярной структурой [1454]. Такие вещества были обнаружены в выхлопных газах автомобилей. Для 80% этих веществ, данные о которых содержались в стандартной базе данных, удалось выявить четыре важные характеристики, причем применение метода линейной регрессии для анализа данных характеристик показало, что этот метод превышает по своей производительности метод ILP. А для оставшихся 20% веществ такие характеристики, отдельно взятые, не позволяют получить надежный прогноз, но система ILP выявила отношения, которые позволили ей превзойти по производительности системы с линейной регрессией, нейронными сетями и деревьями решений. В [798] показано, как обеспечить прогнозирование терапевтической эффективности различных лекарств, исходя из их молекулярной структуры. Анализ этих практических примеров позволяет прийти к выводу, что достижению высокой производительности систем ILP способствует то, что они позволяют представлять новые отношения и использовать фоновые знания. Тот факт, что правила, найденные системами ILP, могут интерпретироваться людьми, стал

причиной того, что эти методы были взяты на вооружение не только специалистами в области компьютерных наук, но и биологами.

Применение методов ILP способствует развитию не только биологии, но и других наук. Одним из наиболее важных таких направлений является обработка естественного языка, в котором системы ILP используются для извлечения сложной реляционной информации из текста. Итоговые результаты, достигнутые в этом научном направлении, описаны в главе 23.

## 19.6. РЕЗЮМЕ

В данной главе рассматривались различные способы, при использовании которых априорные знания могут помочь агенту обучаться на основе новых опытных данных. Поскольку основная часть априорных знаний выражена в терминах реляционных моделей, а не моделей, основанных на атрибутах, в настоящей главе описаны также системы, которые позволяют изучать реляционные модели. Наиболее важные идеи, изложенные в этой главе, перечислены ниже.

- Успешное применение априорных знаний в обучении показывает, что еще более перспективным является **кумулятивное обучение**, в котором обучающиеся агенты повышают свою способность к обучению по мере того, как приобретают все больше и больше знаний.
- Априорные знания способствуют обучению, поскольку позволяют устраниить ошибочные гипотезы, которые в ином случае считались бы совместимыми, а также “дополнять” объяснения примерами, что приводит к созданию более коротких гипотез. Такие возможности часто позволяют добиться ускоренного обучения с помощью меньшего количества примеров.
- Априорные знания могут выполнять различные функции в логическом выводе, а описания этих функций, выраженные в форме **ограничений логического следствия**, позволяют определить целый ряд разнообразных методов обучения.
- В методе **обучения на основе объяснения** (Explanation-Based Learning — EBL) предусматривается извлечение общих правил из отдельных примеров путем формирования объяснений для этих примеров и обобщения объяснений. Метод EBL представляет собой дедуктивный метод, с помощью которого знания основных принципов преобразуются в полезные, эффективные экспертные знания специального назначения.
- В методе **обучения с учетом релевантности** (Relevance-Based Learning — RBL) используются априорные знания в форме определений для выявления релевантных атрибутов, что приводит к созданию уменьшенного пространства гипотез и ускорению обучения. Метод RBL обеспечивает также создание дедуктивных обобщений на основе отдельных примеров.
- В методе **индуктивного обучения на основе знаний** (Knowledge-Based Inductive Learning — KBIL) предусматривается поиск индуктивных гипотез, позволяющих объяснить множество результатов наблюдений с помощью фоновых знаний.
- В методах **индуктивного логического программирования** (Inductive Logic Programming — ILP) метод KBIL применяется к знаниям, выраженным в ло-

тике первого порядка. Методы ILP позволяют получать в процессе обучения такие реляционные знания, которые не могут быть выражены в системах на основе атрибутов.

- Индуктивное логическое программирование может осуществляться в рамках нисходящего подхода, в котором осуществляется уточнение очень общего правила, или восходящего подхода, обратного по отношению к процессу deductive логического вывода.
- С помощью методов ILP естественным путем вырабатываются новые предикаты, с помощью которых могут быть выражены новые емкие теории, поэтому они открывают возможность создания систем формирования научных теорий общего назначения.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

---

Хотя вполне очевидно, что одной из главных тем философии науки должно быть использование априорных знаний в обучении, до последнего времени в этой области был проведен лишь небольшой объем работы. В книге *Fact, Fiction, and Forecast*, написанной философом Нельсоном Гудманом [578], была опровергнута применявшаяся ранее гипотеза о том, что индукция сводится просто к изучению достаточного количества примеров некоторого высказывания с квантором всеобщности, а затем принятия его в качестве гипотезы. Рассмотрим, например, гипотезу “Цвета всех изумрудов изменчивы”, где под ~~и~~ **изменчивым** подразумевается “зеленый, если наблюдения проводятся до времени  $t$ , но голубой, если наблюдения проводятся после этого”. В любое время, предшествующее  $t$ , человеком могли бы рассматриваться миллионы примеров, подтверждающих правило, что цвета изумрудов изменчивы, и ни одного опровергающего примера, и тем не менее не была бы выражена готовность принять это правило. Такое свойство человеческой психики можно объяснить только с помощью анализа той роли, которую играют в процессе индукции релевантные априорные знания. Гудман предложил рассматривать различные виды априорных знаний, которое могли бы оказаться применимыми в логическом выводе, включая одну из версий способа формулировки определений, называемую определением **сверхгипотез**. К сожалению, идеи Гудмана никогда не применялись в исследованиях по машинному обучению.

Истоками подхода к обучения на основе объяснения явились методы, используемые в планировщике Strips [465]. После формирования какого-то плана его обобщенная версия сохраняется в библиотеке планов и используется в дальнейшем планировании в качестве **макрооператора**. Аналогичные идеи просматриваются в архитектуре ACT\* Андерсона, где они выступают под названием **компиляции знаний** [31], и в архитектуре Soar под названием **формирование фрагментов** [881]. Непосредственными причинами быстрого повышения интереса к методам обучения на основе объяснения стало появление методов **приобретения схемы** [379], **аналитического обобщения** [1062] и **обобщения на основе ограничений** [1056], которые были подытожены в статьях [381] и [1065]. Хирш [659] предложил алгоритм EBL, описанный в данной книге, и показал, что этот алгоритм можно непосредственно включить в систему логического программирования. Ван Хармелен и Банди [1531] показали, что метод

обучения на основе объяснения представляет собой один из вариантов метода **частичной оценки**, используемого в системах анализа программ [742].

Проведенный в последнее время строгий анализ привел к лучшему пониманию потенциальных затрат и выгод, связанных с применением метода EBL, с точки зрения его влияния на скорость решения задач. Минтон [1057] показал, что использование метода EBL без значительных дополнительных усилий со стороны пользователя вполне может привести к существенному замедлению работы любой программы. Тамб и др. [1488] обнаружили, что аналогичная проблема возникает при формировании фрагментов знаний, предназначенных для запоминания, и предложили способ уменьшения выразительной мощи языка правил, позволяющий свести к минимуму стоимость операции согласования правил с содержимым рабочей памяти. В этой работе обнаружились явные аналогии с полученными недавно результатами анализа сложности логического вывода в ограниченных версиях логики первого порядка (см. главу 9). Формальный вероятностный анализ ожидаемых преимуществ использования метода EBL можно найти в [595] и [1471]. Превосходный обзор этого метода приведен в [396].

Вместо использования примеров в качестве материала для обобщения их можно применять непосредственно для решения новых задач в процессе, называемом **формированием рассуждений по аналогии**. Такой метод формирования рассуждений осуществляется в нескольких вариантах, во-первых, в виде формирования приемлемых рассуждений с учетом степени подобия [540], во-вторых, в виде дедуктивного логического вывода, основанного на определениях, но требующего привлечения примеров [330], и, в-третьих, в виде метода EBL с “отложенным принятием решений”, который позволяет корректировать направление обобщения старого примера с учетом потребностей решения новой задачи. Последняя разновидность процесса формирования рассуждений по аналогии наиболее часто встречается в системах **формирования рассуждения на основе конкретных случаев** [830] и системах создания **деривационной аналогии** [1540].

Подход к использованию информации о релевантности в форме функциональных зависимостей был впервые разработан в сообществе пользователей баз данных, где он служит для структуризации больших множеств атрибутов на более легко управляемые подмножества. Функциональные зависимости применялись для формирования рассуждений по аналогии Карбонеллом и Коллинзом [222], а их трактовка, более подходящая для использования в логике, была предложена Бобровым и Рафаэлем [144]. Зависимости были повторно открыты независимо и подверглись всестороннему логическому анализу в работах Дэвиса и Рассела [329], [330]. Кроме того, зависимости использовались в методе декларативного смещения Расселом и Грософом [1327]. Эквивалентность таких понятий, как определения и пространства гипотез с ограниченным словарем, была доказана в [1323]. Возможность применения алгоритмов обучения к определениям и достижения повышенной производительности с помощью метода RBDTL впервые была показана на примере алгоритма Focus в [20]. Тадепалли [1485] описал остроумный алгоритм обучения на основе определений, который демонстрирует существенное увеличение скорости обучения.

Идею о том, что индуктивное обучение может осуществляться по методу обратной дедукции, можно проследить до работы В.С. Джевонса [736], который писал: “Изучение формальной логики и теории вероятностей привело меня к тому, что я стал приверженцем мнения, что не существует такой вещи, как отдельный метод ин-

дукции, противопоставляемый дедукции, но индукция является просто обратным воплощением дедукции". Исследования в области вычислительных алгоритмов начались с замечательных тезисов докторской диссертации Гордона Плоткина [1216], опубликованных в Эдинбурге. Хотя Плоткин разработал множество теорем и методов, которые в настоящее время находят применение в индуктивном логическом программировании, он был разочарован некоторыми результатами, показывающими неразрешимость определенных подзадач методами индукции. В системе MIS [1396] снова была предпринята попытка решения задачи изучения логических программ, но самим автором эта работа в основном рассматривалась как вклад в теорию автоматизированной отладки. Работа в области индуктивного вывода правил, подобная проведенной при создании систем ID3 [1257] и CN2 [263], привела к появлению системы Foil [1258], которая впервые позволила осуществлять на практике методы индуктивного вывода реляционных правил. Исследования в области реляционного обучения получили новый стимул после публикации работы Магглтона и Бантайна [1100], в которой была описана программа Cigol, реализующая немного неполную версию метода обратной резолюции, но способную вырабатывать новые предикаты<sup>5</sup>. Задачи изобретения предикатов рассматриваются также в [1604]. Следующей важной системой стала Golem [1102], в которой используется алгоритм формирования покрытия, основанный на выдвинутом Плоткином понятии относительного наименьшего общего обобщения. Тогда как система Foil была нисходящей, системы Cigol и Golem действовали в восходящем направлении. К примерам других систем, появившихся в тот период, относятся Itou [1312] и Clint [352]. В разработанной в дальнейшем системе Progol [1098] был принят гибридный (нисходящий и восходящий) подход к формированию обратного логического следствия; эта система применялась для решения целого ряда практических задач, особенно в области биологии и обработки естественного языка. В [1099] описано расширение системы Progol, позволяющее учитывать неопределенность, в форме стохастических логических программ.

Формальный анализ методов ILP приведен в [1096], большая подборка статей представлена в [1097], а описания методов и приложений собраны в [897]. В [1163] можно найти более современный обзор истории развития этой области и описание проблем, которые должны быть решены в будущем. Первые результаты анализа сложности, опубликованные в [632], свидетельствовали о том, что задача изучения высказываний в логике первого порядка является безнадежно сложной. Однако благодаря лучшему пониманию важности синтаксических ограничений различного рода, налагаемых на выражения, удалось получить положительные результаты даже для выражений с рекурсией [426]. Результаты изучения возможности обучения для методов ILP изложены в [279] и [794].

Несмотря на то что в настоящее время индуктивное логическое программирование, по-видимому, является доминирующим подходом к решению задач конструктивной индукции, это — не единственный опробованный подход. Разработаны так называемые **системы открытий**, которые направлены на моделирование процесса научного открытия новых понятий, обычно путем прямого поиска в пространстве определений понятий. В системе AM, или Automated Mathematician (автоматизированный матема-

---

<sup>5</sup> Метод обратной резолюции опубликован также в [1322], где в одной из сносок приведен простой алгоритм.

тик), Дуга Лената [337] использовались эвристики открытий, выраженные в виде правил экспертной системы для управления осуществляемым в ней поиском понятий и научных предположений в области элементарной теории чисел. В отличие от большинства систем, предназначенных для формирования математических рассуждений, в системе AM отсутствовало понятие доказательства, поэтому она могла выдвигать только предположения. Эта система повторно открыла предположение Гольдбаха и теорему уникального разложения на простые множители. Архитектура системы AM была обобщена в системе Eurisko [907] в результате введения механизма, способного перезаписывать собственные эвристики открытия этой системы. Система Eurisko применялась во многих областях, отличных от области поиска математических открытий, хотя и с меньшим успехом, чем AM. Но методология систем AM и Eurisko оказалась противоречивой [909], [1292].

Еще один класс систем открытия предназначен для обработки реальных научных данных в целях поиска новых законов. Системы Dalton, Glauber и Stahl [885] представляют собой системы на основе правил, предназначенные для поиска количественных связей в экспериментальных данных, полученных из физических систем; в каждом случае системы показали свою способность снова выдвинуть широко известное открытие из истории науки. Системы открытия, основанные на вероятностных методах (в частности, алгоритмы кластеризации, позволяющие обнаруживать новые категории), рассматриваются в главе 20.

## УПРАЖНЕНИЯ

- 19.1.** Преобразовав выражение в конъюнктивную нормальную форму и применив метод резолюции, покажите, что приведенное на с. 923 заключение, касающееся бразильцев, является обоснованным.
- 19.2.** Для каждого из следующих определений запишите логическое представление и объясните, почему это определение является истинным (если его действительно можно считать таковым).
  - a)** Название штата (в США) можно определить по почтовому индексу.
  - б)** Масса монеты зависит от ее проекта и номинала.
  - в)** Применительно к конкретной программе выходные данные определяются входными данными.
  - г)** Изменение массы тела в большую или меньшую сторону происходит под влиянием климата, рациона питания, физической нагрузки и интенсивности обмена веществ.
  - д)** Наличие лысины у мужчины определяется тем, была ли лысина у деда со стороны матери.
- 19.3.** Могла бы найти применение вероятностная версия определения? Предложите приемлемую трактовку такого понятия.
- 19.4.** Подставьте недостающие значения выражения  $C_1$  или  $C_2$  (или обоих выражений) в приведенный ниже ряд выражений, при условии, что  $C$  — резольвента  $C_1$  и  $C_2$ .
  - а)**  $C = \text{True} \Rightarrow P(A, B), C_1 = P(x, y) \Rightarrow Q(x, y), C_2 = ??.$
  - б)**  $C = \text{True} \Rightarrow P(A, B), C_1 = ??, C_2 = ??.$

**в)**  $C = P(x, y) \Rightarrow P(x, f(y)), C_1 = ??, C_2 = ??.$

Если имеется больше одного возможного решения, приведите по одному примеру каждого из различных решений.

- 19.5.** Предположим, что разрабатывается логическая программа, которая осуществляет один из этапов логического вывода по методу резолюции. Это означает, что такая программа,  $\text{Resolve}(c_1, c_2, c)$ , завершается успешно, если  $c$  — результат применения операции резолюции к выражениям  $c_1$  и  $c_2$ . При обычных обстоятельствах программа  $\text{Resolve}$  используется в составе системы автоматического доказательства теорем, в которой она вызывается с параметрами  $c_1$  и  $c_2$ , конкретизированными значениями некоторых выражений, и возвращает значение резольвенты  $c$ . А теперь предположим, что вместо этого данная программа вызывается с конкретизированным значением  $c$  и неконкретизированными значениями  $c_1$  и  $c_2$ . Приведет ли это к успешному получению приемлемых результатов на этапе обратной резолюции? Потребуется ли внести какие-либо специальные изменения в рассматриваемую систему логического программирования для того, чтобы можно было применять программу в такой форме?
- 19.6.** Предположим, что в системе  $\text{Foil}$  рассматривается задача введения литерала в выражение с использованием бинарного предиката  $P$  и что предыдущие литералы (включая голову выражения) содержат пять разных переменных.
- Какое количество функционально различных литералов может быть сформировано? Два литерала являются функционально идентичными, если они отличаются только по именам содержащихся в них новых переменных.
  - Можете ли вы найти общую формулу для количества различных литералов с предикатом, имеющим арность  $x$ , если ранее использовалось  $n$  переменных?
  - Почему в системе  $\text{Foil}$  не допускается введение литералов, которые не содержат ранее использованных переменных?
- 19.7.** Используя данные из генеалогического дерева, приведенного на рис. 19.8, или подмножество этих данных, примените алгоритм  $\text{Foil}$  для изучения определения предиката  $\text{Ancestor}$ .

# 20 СТАТИСТИЧЕСКИЕ МЕТОДЫ ОБУЧЕНИЯ

*В данной главе рассматривается обучение как способ формирования рассуждений в условиях неопределенности на основании результатов наблюдений.*

В главах части V было указано, что в реальных вариантах среды ситуации преимущественно являются неопределенными. Агенты могут справиться с неопределенностью, используя методы теории вероятностей и теории решений, но вначале должны сформировать в процессе обучения на основании полученного опыта свои вероятностные теории о мире. В этой главе показано, как может быть достигнута эта цель. В ней описано, как сформулировать саму задачу обучения в виде процесса вероятностного вывода (раздел 20.1). Материал, изложенный в данной главе, свидетельствует о том, что байесовский подход к обучению является чрезвычайно мощным и предоставляет общие решения проблем шума, чрезмерно тщательной подгонки и оптимального предсказания. В настоящей главе учитывается также тот факт, что агент, не являющийся полностью всезнающим, не будет никогда уверен в том, что та или иная теория о мире действительно является правильной, но все равно обязан принимать решения, используя некоторую теорию о мире.

В разделах 20.2 и 20.3 описаны методы обучения вероятностных моделей (в основном байесовских сетей). В разделе 20.4 рассматриваются методы обучения, предусматривающие сохранение и извлечение из памяти конкретных экземпляров примеров. В разделе 20.5 описано обучение **нейронных сетей**, а в разделе 20.6 даны вводные сведения о **ядерных машинах**. Часть материала этой главы имеет богатое математическое содержание (и для ее освоения требуются элементарные знания в области многомерного исчисления), хотя основные приведенные здесь научные результаты можно понять, не углубляясь в детали. При изучении этого материала читателю может потребоваться еще раз просмотреть главы 13 и 14 и ознакомиться с математическими сведениями, приведенными в приложении A.

## 20.1. СТАТИСТИЧЕСКОЕ ОБУЧЕНИЕ

Основными понятиями в данной главе, как и в главе 18, являются **данные** и **гипотезы**. Но в этой главе данные рассматриваются как **свидетельства**, т.е. конкретизации некоторых или всех случайных переменных, описывающих проблемную область, а гипотезы представляют собой вероятностные теории того, как функционирует проблемная область, включающие логические теории в качестве частного случая.

Рассмотрим очень простой пример. Наши любимые леденцы “Сюрприз” выпускаются в двух разновидностях: вишневые (сладкие) и лимонные (кислые). У изготавителя леденцов особое чувство юмора, поэтому он заворачивает каждую конфету в одинаковую непрозрачную бумагу, независимо от разновидности. Леденцы продаются в очень больших пакетах (также внешне не различимых), о которых известно, что они относятся к пяти следующим типам:

- $h_1$ : 100% вишневых леденцов
- $h_2$ : 75% вишневых + 25 % лимонных леденцов
- $h_3$ : 50% вишневых + 50 % лимонных леденцов
- $h_4$ : 25% вишневых + 75 % лимонных леденцов
- $h_5$ : 100 % лимонных леденцов

Получив новый пакет леденцов, любитель конфет пытается угадать, к какому типу он относится, и обозначает тип пакета случайной переменной  $H$  (сокращение от hypothesis — гипотеза), которая имеет возможные значения от  $h_1$  до  $h_5$ . Безусловно, значение переменной  $H$  невозможно определить с помощью непосредственного наблюдения. По мере развертывания и осмотра конфет регистрируются данные о них,  $D_1, D_2, \dots, D_N$ , где каждый элемент данных,  $D_i$ , представляет собой случайную переменную с возможными значениями *cherry* (вишневый леденец) и *lime* (лимонный леденец). Основная задача, стоящая перед агентом, состоит в том, что он должен предсказать, к какой разновидности относится следующая конфета<sup>1</sup>. Несмотря на кажущуюся простоту, постановка этой задачи позволяет ознакомиться с многими важными темами. В действительности агент должен вывести логическим путем теорию о мире, в котором он существует, хотя и очень простую.

В **байесовском обучении** исходя из полученных данных просто вычисляется вероятность каждой гипотезы и на этой основе делаются предсказания. Это означает, что предсказания составляются с использованием всех гипотез, взвешенных по их вероятностям, а не с применением только одной “наилучшей” гипотезы. Таким образом, обучение сводится к вероятностному выводу. Допустим, что переменная  $\mathbf{D}$  представляет все данные, с наблюдаемым значением  $\mathbf{d}$ ; в таком случае вероятность каждой гипотезы может быть определена с помощью правила Байеса:

$$P(h_i | \mathbf{d}) = \alpha P(\mathbf{d} | h_i) P(h_i) \quad (20.1)$$

Теперь предположим, что необходимо сделать предсказание в отношении неизвестного количества  $x$ . В таком случае применяется следующее уравнение:

<sup>1</sup> Читатели, искушенные в статистике, узнают в этом сценарии один из вариантов постановки задачи с **урнами и шарами**. Но авторы считают урны и шары менее привлекательными, чем леденцы; более того, с самими леденцами можно также связать и другие задачи, например задачу принятия решения о том, есть ли смысл перепродать какой-то пакет конфет своему другу (см. упр. 20.3).

$$\begin{aligned}
 \mathbf{P}(X|\mathbf{d}) &= \sum_i \mathbf{P}(X|\mathbf{d}, h_i) \mathbf{P}(h_i|\mathbf{d}) \\
 &= \sum_i \mathbf{P}(X|h_i) P(h_i|\mathbf{d})
 \end{aligned} \tag{20.2}$$

где предполагается, что каждая гипотеза определяет распределение вероятностей по  $X$ . Это уравнение показывает, что предсказания представляют собой взвешенные средние по предсказаниям отдельных гипотез. Самы гипотезы по сути являются “посредниками” между фактическими данными и предсказаниями. Основными количественными показателями в байесовском подходе являются  $\bowtie$  **распределение априорных вероятностей гипотезы**,  $P(h_i)$ , и  $\bowtie$  **правдоподобие** данных согласно каждой гипотезе,  $P(\mathbf{d}|h_i)$ .

Применимельно к рассматриваемому примеру с леденцами предположим, что изготовитель объявил о наличии распределения априорных вероятностей по значениям  $h_1, \dots, h_5$ , которое задано вектором  $<0.1, 0.2, 0.4, 0.2, 0.1>$ . Правдоподобие данных рассчитывается в соответствии с предположением, что наблюдения характеризуются свойством  $\bowtie$  **i.i.d.**, т.е. являются независимыми и одинаково распределенными (i.i.d. — independently and identically distributed), поэтому соблюдается следующее уравнение:

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i) \tag{20.3}$$

Например, предположим, что пакет в действительности представляет собой пакет такого типа, который состоит из одних лимонных леденцов ( $h_5$ ), и все первые 10 конфет являются лимонными леденцами; в таком случае значение  $P(\mathbf{d}|h_3)$  равно  $0.5^{10}$ , поскольку в пакете типа  $h_3$  половина конфет — лимонные леденцы<sup>2</sup>. На рис. 20.1, *a* показано, как изменяются апостериорные вероятности пяти гипотез по мере наблюдения последовательности из 10 лимонных леденцов. Обратите внимание на то, что кривые вероятностей начинаются с их априорных значений, поэтому первоначально наиболее вероятным вариантом является гипотеза  $h_3$  и остается такойой после развертывания 1 конфеты с лимонным леденцом. После развертывания 2 конфет с лимонными леденцами наиболее вероятной становится гипотеза  $h_4$ , а после обнаружения 3 или больше лимонных леденцов наиболее вероятной становится гипотеза  $h_5$  (ненавистный пакет, состоящий из одних кислых лимонных леденцов). После обнаружения 10 подряд лимонных леденцов мы почти уверены в своей злосчастной судьбе. На рис. 20.1, *b* приведена предсказанная вероятность того, что следующий леденец будет лимонным, согласно уравнению 20.2. Как и следовало ожидать, она монотонно увеличивается до 1.

<sup>2</sup> Выше было указано, что пакеты с конфетами — очень большие, так как в противном случае предположение i.i.d. не соблюдалось бы. Формально было бы более правильно (но менее гигиенично) снова заворачивать каждую конфету в бумагу после ее осмотра и возвращать в пакет.

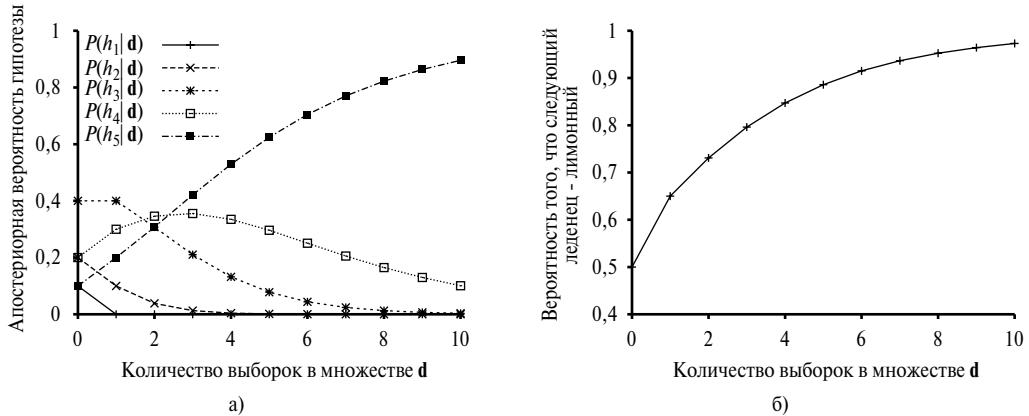


Рис. 20.1. Изменение вероятностей в зависимости от количества данных: апостериорные вероятности  $P(h_i | d_1, \dots, d_N)$ , полученные с помощью уравнения 20.1. Количество наблюдений  $N$  возрастает от 1 до 10, а в каждом наблюдении обнаруживается лимонный леденец (а); байесовские предсказания  $P(d_{N+1} = \text{lime} | d_1, \dots, d_N)$ , полученные из уравнения 20.2 (б)

Этот пример показывает, что  $\text{---}$  истинная гипотеза в конечном итоге будет доминировать над байесовским предсказанием. В этом состоит характерная особенность байесовского обучения. При любом заданном распределении априорных вероятностей, которое не исключает с самого начала истинную гипотезу, апостериорная вероятность любой ложной гипотезы в конечном итоге полностью исчезает просто потому, что вероятность неопределенно долгого формирования “некартерных” данных исчезающе мала (сравните это замечание с аналогичным замечанием, сделанным при обсуждении РАС-обучения в главе 18). Еще более важно то, что байесовское предсказание является оптимальным, независимо от того, применяется ли большой или малый набор данных. При наличии распределения априорных вероятностей гипотезы все другие предсказания будут правильными менее часто.

Но за оптимальность байесовского обучения, безусловно, приходится платить. В реальных задачах обучения пространство гипотез обычно является очень большим или бесконечным, как было показано в главе 18. В некоторых случаях операция вычисления суммы в уравнении 20.2 (или, в непрерывном случае, операция интегрирования) может быть выполнена успешно, но в большинстве случаев приходится прибегать к приближенным или упрощенным методам.

Один из широко распространенных приближенных подходов (из числа тех, которые обычно применяются в научных исследованиях) состоит в том, чтобы делать предсказания на основе единственной наиболее вероятной гипотезы, т.е. той гипотезы  $h_1$ , которая максимизирует значение  $P(h_1 | \mathbf{d})$ . Такую гипотезу часто называют  $\text{---}$  максимальной апостериорной гипотезой, или сокращенно MAP (Maximum A Posteriori; произносится “эм-эй-пи”). Предсказания  $h_{\text{MAP}}$ , сделанные на основе MAP-гипотезы, являются приближенно байесовскими до такой степени, что  $P(X | \mathbf{d}) \approx P(X | h_{\text{MAP}})$ . В рассматриваемом примере с конфетами  $h_{\text{MAP}} = h_5$  после обнаружения трех лимонных леденцов подряд, поэтому агент, обучающийся с помощью MAP-гипотезы, после этого предсказывает, что четвертая конфета представляет собой лимонный леденец, с вероятностью 1.0, а это — гораздо более радикальное предсказание, чем байесовское предсказание вероятности 0.8, приведенное на рис. 20.1.

По мере поступления дополнительных данных предсказания с помощью МАР-гипотезы и байесовские предсказания сближаются, поскольку появление гипотез, конкурирующих с МАР-гипотезой, становится все менее и менее вероятным. Хотя в рассматриваемом примере это не показано, поиск МАР-гипотез часто бывает намного проще по сравнению с байесовским обучением, поскольку требует решения задачи оптимизации, а не задачи вычисления большой суммы (или интегрирования). Примеры, подтверждающие это замечание, будут приведены ниже в данной главе.

И в байесовском обучении, и в обучении с помощью МАР-гипотез важную роль играет распределение априорных вероятностей гипотезы  $P(h_i)$ . Как было показано в главе 18, если пространство гипотез является слишком выразительным, в том смысле, что содержит много гипотез, хорошо согласующихся с набором данных, то может происходить **чрезмерно тщательная подгонка**. С другой стороны, байесовские методы обучения и методы обучения на основе МАР-гипотез не налагают произвольный предел на количество подлежащих рассмотрению гипотез, а позволяют использовать распределение априорных вероятностей для наложения штрафа за сложность. Как правило, более сложные гипотезы имеют более низкую априорную вероятность, отчасти потому, что сложных гипотез обычно бывает намного больше, чем простых. С другой стороны, более сложные гипотезы имеют большую способность согласовываться с данными (в крайнем случае какая-то поисковая таблица может оказаться способной точно воспроизводить данные с вероятностью 1). Поэтому в распределении априорных вероятностей гипотезы воплощен компромисс между сложностью гипотезы и степенью ее согласования с данными.

Влияние такого компромисса можно наблюдать наиболее наглядно в случае использования логических гипотез, когда переменная  $H$  содержит только детерминированные гипотезы. В таком случае значение  $P(\mathbf{a} | h_i)$  равно 1, если гипотеза  $h_i$  согласуется с данными, и 0 — в противном случае. Рассматривая уравнение 20.1, можно определить, что  $h_{\text{MAP}}$  в таких условиях представляет собой *простейшую логическую теорию, согласованную с данными*. Поэтому обучение с помощью максимальной апостериорной гипотезы представляет собой естественное воплощение принципа бритвы Оккама.

Еще один способ анализа компромисса между сложностью и степенью согласованности состоит в том, что можно исследовать уравнение 20.1, взяв его логарифм. Применение значения  $h_{\text{MAP}}$  для максимизации выражения  $P(\mathbf{a} | h_i) P(h_i)$  эквивалентно минимизации следующего выражения:

$$-\log_2 P(\mathbf{a} | h_i) - \log_2 P(h_i)$$

Используя связь между информационным содержанием и вероятностью, которая была описана в главе 18, можно определить, что терм  $-\log_2 P(h_i)$  определяет количество битов, требуемых для задания гипотезы  $h_i$ . Кроме того, терм  $-\log_2 P(\mathbf{a} | h_i)$  представляет собой дополнительное количество битов, требуемых для задания данных, если дана рассматриваемая гипотеза (чтобы убедиться в этом, достаточно отметить, что если гипотеза точно предсказывает данные, как в случае гипотезы  $h_5$  и сплошного ряда конфет с лимонными леденцами, не требуется ни одного бита, поскольку  $\log_2 1 = 0$ ). Таким образом, обучение с помощью МАР-гипотезы равносильно выбору гипотезы, которая обеспечивает максимальное сжатие данных. Такую же задачу можно решить более прямо с помощью метода обучения на основе **минимальной длины описания**, или сокращенно MDL (Minimum Description

Length), в котором вместо манипуляций с вероятностями предпринимаются попытки минимизировать размер гипотезы и закодированного представления данных.

Окончательное упрощение может быть достигнуто путем принятия предположения о **равномерном** распределении априорных вероятностей по пространству гипотез. В этом случае обучение с помощью МАР-гипотезы сводится в выбору гипотезы  $h_i$ , которая максимизирует значение  $P(\mathbf{d} | H_i)$ . Такая гипотеза называется гипотезой с **максимальным правдоподобием** (Maximum Likelihood — ML) и сокращенно обозначается  $h_{ML}$ . Обучение на основе гипотезы с максимальным правдоподобием очень широко применяется в статистике, поскольку в этой научной области многие исследователи не доверяют распределениям априорных вероятностей гипотезы, считая, что они имеют субъективный характер. Это — приемлемый подход, применяемый в тех обстоятельствах, когда нет оснований априорно отдавать предпочтение одной гипотезе перед другой, например, в тех условиях, когда все гипотезы являются в равной степени сложными. Такой метод обучения становится хорошей аппроксимацией байесовского обучения и обучения с помощью МАР-гипотезы, когда набор данных имеет большие размеры, поскольку данные сами исправляют распределение априорных вероятностей по гипотезам, но связан с возникновением определенных проблем (как будет показано ниже) при использовании небольших наборов данных.

## 20.2. ОБУЧЕНИЕ С ПОМОЩЬЮ ПОЛНЫХ ДАННЫХ

Начнем разработку методов статистического обучения с простейшей задачи — **обучение параметрам** с помощью **полных данных**. Задача обучения параметрам сводится к поиску числовых параметров для вероятностной модели, имеющей фиксированную структуру. Например, может потребоваться определить в процессе обучения условные вероятности в байесовской сети с заданной структурой. Данные называются **полными**, если каждая точка данных содержит значения для каждой переменной в вероятностной модели, применяемой при обучении. При наличии полных данных задача определения в процессе обучения параметров сложной модели значительно упрощается. Кроме того, в данном разделе кратко рассматривается задача изучения структуры.

### Обучение параметрам с помощью метода максимального правдоподобия: дискретные модели

Допустим, что мы покупаем пакет конфет с лимонными и вишневыми леденцами, выпущенный новым изготовителем, соотношение лимонных и вишневых леденцов в продукции которого полностью неизвестно; это означает, что доля тех и других леденцов может измеряться любым значением от 0 до 1. В данном случае приходится рассматривать континuum гипотез. Кроме того, в этом случае **параметром**, который будет обозначаться как  $\theta$ , является доля вишневых леденцов, а гипотезой является  $h_\theta$  (доля лимонных леденцов выражается как  $1-\theta$ ). Если принято предположение, что все возможные значения долевого состава априорно являются равновероятными, то становится обоснованным подход на основе гипотезы с максимальным правдоподобием. Если мы промоделируем эту ситуацию с помо-

шью байесовской сети, то потребуется только одна случайная переменная, *Flavor* (разновидность конфеты, случайно выбранной из пакета). Эта переменная принимает значения *cherry* и *lime*, где вероятность *cherry* равна  $\theta$  (рис. 20.2, *a*). Теперь предположим, что развернуто  $N$  конфет, из которых  $c$  оказались вишневыми леденцами, а  $\ell = N - c$  были лимонными леденцами. Согласно уравнению 20.3, правдоподобие этого конкретного набора данных выражается следующей формулой:

$$P(\mathbf{d} | h_{\theta}) = \prod_{j=1}^N P(d_j | h_{\theta}) = \theta^c \cdot (1-\theta)^\ell$$

Гипотеза с максимальным правдоподобием задается значением  $\theta$ , которое максимизирует это выражение. Такое же значение может быть получено путем максимизации значения **логарифмического правдоподобия**:

$$L(\mathbf{d} | h_{\theta}) = \log P(\mathbf{d} | h_{\theta}) = \sum_{j=1}^N \log P(d_j | h_{\theta}) = c \log \theta + \ell \log (1-\theta)$$

(Взяв логарифмы, мы преобразовали произведение в сумму по данным, которую обычно легче максимизировать.) Чтобы найти значение максимального правдоподобия  $\theta$ , дифференцируем  $L$  по  $\theta$  и приравняем полученное выражение к нулю следующим образом:

$$\frac{dL(\mathbf{d} | h_{\theta})}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

Таким образом, если описать это выражение на естественном языке, то гипотеза с максимальным правдоподобием  $h_{ML}$  утверждает, что фактическая доля вишневых леденцов в пакете равна наблюдаемой доле этих леденцов в конфетах, развернутых до сих пор!

На первый взгляд создается впечатление, что мы проделали большой объем работы лишь для того, чтобы открыть этот очевидный факт. Но в действительности описаным выше путем был создан один из стандартных методов обучения параметрам с максимальным правдоподобием, который описан ниже.

1. Записать выражение для правдоподобия данных как функции от параметра (параметров).
2. Найти производную логарифмического правдоподобия по отношению к каждому параметру.
3. Найти такие значения параметров, чтобы производные стали равными нулю.

Обычно самым сложным является последний из этих этапов. В рассматриваемом примере он был простейшим, но ниже будет показано, что во многих случаях приходится прибегать к использованию итерационных алгоритмов поиска решений или других числовых методов оптимизации, как было описано в главе 4. Кроме того, этот пример иллюстрирует важную проблему, которая в целом характерна для обучения с учетом максимального правдоподобия: *если набор данных достаточно мал и поэтому некоторые события еще не наблюдались (например, не было обнаружено ни*

одной конфеты с вишневым леденцом), то гипотеза с максимальным правдоподобием присваивает этим событиям нулевую вероятность. Для предотвращения возникновения этой проблемы использовались различные приемы, такие как инициализация счетчиков для каждого события значением 1, а не 0.

Рассмотрим еще один пример. Предположим, что этот новый изготовитель конфет хочет дать потребителю небольшую подсказку и использует для конфет обертки красного и зеленого цветов. Значение переменной  $Wrapper$ , соответствующей цвету обертки для каждой конфеты, выбирается по вероятностным законам, в соответствии с некоторым неизвестным условным распределением, но в зависимости от разновидностей конфет. Соответствующая вероятностная модель показана на рис. 20.2, б. Обратите внимание на то, что она имеет три параметра:  $\theta$ ,  $\theta_1$  и  $\theta_2$ . С использованием этих параметров правдоподобие события, связанного, скажем, с обнаружением вишневого леденца в зеленой обертке, можно определить на основе стандартной семантики для байесовских сетей (с. 664):

$$\begin{aligned} P(Flavor=cherry, Wrapper=green | h_{\theta, \theta_1, \theta_2}) \\ = P(Flavor=cherry | h_{\theta, \theta_1, \theta_2}) P(Wrapper=green | Flavor=cherry, h_{\theta, \theta_1, \theta_2}) \\ = \theta \cdot (1-\theta_1) \end{aligned}$$

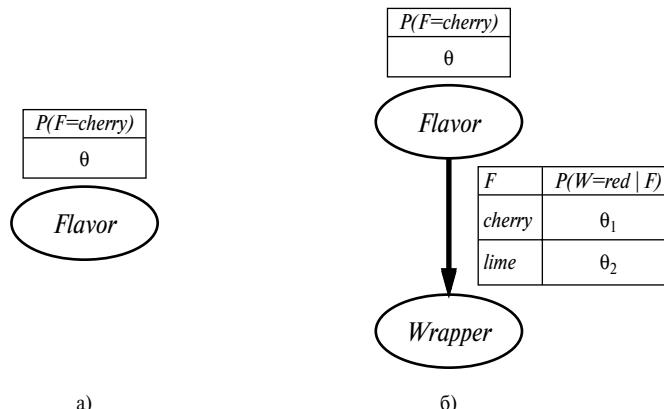


Рис. 20.2. Обучение параметром с помощью байесовской сети: модель в виде байесовской сети для случая, в котором доля вишневых и лимонных леденцов в пакете неизвестна (а); модель для того случая, когда цвета обертки связаны (вероятностной) зависимостью с разновидностями конфет (б)

Теперь допустим, что развернуто  $N$  конфет, из которых  $c$  оказались вишневыми леденцами, а  $\ell$  — лимонными, а количество оберток оказалось таковым:  $x_c$  вишневых леденцов имели красные обертки, а  $g_c$  — зеленые, тогда как  $x_\ell$  лимонных леденцов имели красные обертки, а  $g_\ell$  — зеленые. Правдоподобие этих данных выражается следующим образом:

$$P(\mathbf{d} | h_{\theta, \theta_1, \theta_2}) = \theta^c (1-\theta)^{\ell} \cdot \theta_{\text{F}}^{x_c} (1-\theta_1)^{g_c} \cdot \theta_2^{x_\ell} (1-\theta_2)^{g_\ell}$$

На первый взгляд это соотношение кажется весьма сложным, но его можно упростить, взяв логарифмы, следующим образом:

$$L = [c \log \theta + \ell \log (1-\theta)] + [r_c \log \theta_1 + g_c \log (1-\theta_1)] + [r_g \log \theta_2 + g_g \log (1-\theta_2)]$$

Преимущество взятия логарифмов является очевидным — логарифмическое правдоподобие представляет собой сумму трех термов, каждый из которых содержит единственный параметр. После взятия производных по каждому параметру и приравнивания их к нулю будет получено три независимых уравнения, каждое из которых содержит только один параметр:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+\ell}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \Rightarrow \theta_1 = \frac{r_c}{r_c+g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_g}{\theta_2} - \frac{g_g}{1-\theta_2} = 0 \Rightarrow \theta_2 = \frac{r_g}{r_g+g_g}$$

Решение для  $\theta$  остается таким же, как и прежде. Решение для  $\theta_1$ , вероятности того, что вишневый леденец имеет красную обертку, представляет собой наблюдаемую долю вишневых леденцов в красных обертках, и аналогичным образом определяется решение для  $\theta_2$ .

Эти результаты являются очень удобными, и легко показать, что их можно распространить на любую байесовскую сеть, условные вероятности в которой представлены в виде таблицы. Наиболее важный вывод состоит в том, что ~~если~~ при наличии полных данных задача обучения параметрам с максимальным правдоподобием для байесовской сети декомпонуется на отдельные задачи обучения, по одной для каждого параметра<sup>3</sup>. Еще один важный вывод состоит в том, что значения параметра для любой переменной при наличии ее родительских значений представляют собой наблюдаемые частоты значений переменных для каждого набора родительских значений. Как и прежде, необходимо внимательно следить за предотвращением появления нулевых значений, если набор данных является небольшим.

## Наивные байесовские модели

По-видимому, к числу моделей на основе байесовской сети, которые наиболее широко используются в машинном обучении, относятся **наивные байесовские** модели. В таких моделях переменная “класса”  $C$  (значение которой должно быть предсказано) задана в корневом узле, а переменные “атрибутов”  $X_i$  заданы в листовых узлах. Такие модели называются “наивными”, поскольку в них предполагается, что атрибуты являются условно независимыми друг от друга, если определен рассматриваемый класс (модель, приведенная на рис. 20.2, б, представляет собой наивную байесовскую модель только с одним атрибутом). При условии, что переменные являются булевыми, рассматриваемые параметры принимают такой вид:

$$\theta = P(C=true), \quad \theta_{i1} = P(X_i=true | C=true), \quad \theta_{i2} = P(X_i=true | C=false)$$

Значения параметров с максимальным правдоподобием можно найти с помощью точно такого же способа, который применялся в сети на рис. 20.2, б. Сразу после

<sup>3</sup> Случай, в котором отсутствуют таблицы условных вероятностей и каждый параметр влияет на несколько условных вероятностей, рассматривается в упр. 20.7.

обучения данной модели с помощью такого способа она может использоваться для классификации новых примеров, в которых переменная класса  $C$  является ненаблюдаемой. При наличии значений наблюдаемых атрибутов  $x_1, \dots, x_n$  вероятность каждого класса определяется следующим соотношением:

$$\mathbf{P}(C|x_1, \dots, x_n) = \alpha \cdot \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C)$$

Детерминистическое предсказание может быть получено путем выбора наиболее вероятного класса. На рис. 20.3 показана кривая обучения для этого метода, соответствующая примеру его применения к задаче с рестораном, описанной в главе 18. Обучение с помощью этого метода происходит довольно успешно, но не так хорошо, как при обучении деревьев решений; следует полагать, что связано с тем, что истинная гипотеза (представляющая собой дерево решений) не является точно представимой с помощью наивной байесовской модели. Как оказалось, метод наивного байесовского обучения действует удивительно успешно в самых разнообразных приложениях, а его усиленная версия (упр. 20.5) является одним из наиболее эффективных алгоритмов обучения общего назначения. Метод наивного байесовского обучения хорошо масштабируется на очень большие задачи: при наличии  $n$  булевых атрибутов имеется только  $2n+1$  параметров и ~~для обнаружения наивной байесовской гипотезы с максимальным правдоподобием,  $hML$ , не требуется поиск~~. Наконец, метод наивного байесовского обучения не сталкивается с затруднениями при обработке зашумленных данных и может предоставить вероятностные предсказания, когда это необходимо.



Рис. 20.3. Кривая обучения для случая применения метода наивного байесовского обучения к задаче с рестораном из главы 18; для сравнения показана кривая обучения для случая применения метода обучения дерева решений

### Обучение параметрам с максимальным правдоподобием: непрерывные модели

Непрерывные вероятностные модели, такие как **линейная гауссова** модель, описывались в разделе 14.3. Поскольку в реальных приложениях в основном используются непрерывные переменные, важно знать, как должно осуществляться обучение

непрерывных моделей на основе данных. Принципы обучения с максимальным правдоподобием идентичны применяемым в дискретном случае.

Начнем с очень простого случая: обучение параметрам гауссовой функции плотности от одной переменной. Это означает, что данные вырабатываются следующим образом:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Параметрами этой модели являются математическое ожидание  $\mu$  и среднеквадратичное отклонение  $\sigma$  ( обратите внимание на то, что нормализующая “константа” зависит от  $\sigma$ , поэтому ее нельзя игнорировать). Допустим, что наблюдаемыми значениями являются  $x_1, \dots, x_N$ . В таком случае логарифмическое правдоподобие определяется следующим образом:

$$L = \sum_{j=1}^N \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^N \frac{(x_j-\mu)^2}{2\sigma^2}$$

Приравняв, как обычно, производные к нулю, получим такие уравнения:

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2} \sum_{j=1}^N (x_j - \mu) = 0 \Rightarrow \mu = \frac{\sum_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^N (x_j - \mu)^2 = 0 \Rightarrow \\ \sigma &= \sqrt{\frac{\sum_j (x_j - \mu)^2}{N}} \end{aligned} \quad (20.4)$$

Таким образом, значение максимального правдоподобия среднего представляет собой среднее по выборкам, а значение максимального правдоподобия среднеквадратичного отклонения выражается квадратным корнем от дисперсии выборки. И в данном случае получены удобные результаты, которые подтверждают обоснованность практических методов, созданных на основе “здравого смысла”.

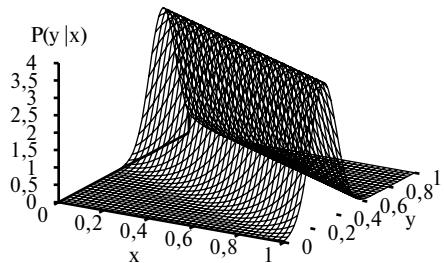
Теперь рассмотрим линейную гауссову модель с одним непрерывным родительским значением  $X$  и непрерывным дочерним значением  $Y$ . Как было описано на стр. 672, значение  $Y$  имеет гауссово распределение, математическое ожидание которого линейно зависит от значения  $X$ , а среднеквадратичное отклонение является постоянным. Чтобы определить в результате обучения распределение условных вероятностей  $P(Y|X)$ , можно максимизировать условное правдоподобие следующим образом:

$$P(Y|X) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y - (\theta_1x + \theta_2))^2}{2\sigma^2}} \quad (20.5)$$

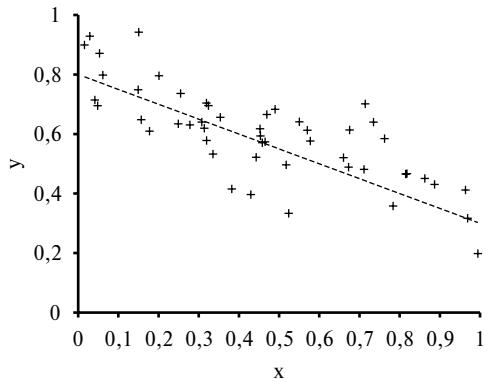
где параметрами являются  $\theta_1$ ,  $\theta_2$  и  $\sigma$ . Данные представляют собой множество пар  $(x_j, y_j)$ , как показано на рис. 20.4. Используя обычные методы (упр. 20.6), можно

найти значения параметров с максимальным правдоподобием. Но в этом контексте нужно сделать еще одно замечание. Если рассматриваются только параметры  $\theta_1$  и  $\theta_2$ , которые определяют линейную связь между  $x$  и  $y$ , то становится очевидно, что максимизация логарифмического правдоподобия по отношению к этим параметрам равносильна минимизации числителя в экспоненте уравнения 20.5:

$$E = \sum_{j=1}^N (y_j - (\theta_1 x_j + \theta_2))^2$$



a)



б)

Рис. 20.4. Примеры применения линейной гауссовой модели: линейная гауссова модель, описанная как  $y = \theta_1 x + \theta_2$ , к которой добавляется гауссов шум с постоянной дисперсией (а); множество из 50 точек данных, сформированных с помощью этой модели (б)

Величина  $(y_j - (\theta_1 x_j + \theta_2))$  представляет собой ~~ошибку~~ для  $(x_j, y_j)$ , т.е. разность между фактическим значением  $y_j$  и прогнозируемым значением  $(\theta_1 x_j + \theta_2)$ , поэтому  $E$  представляет собой хорошо известную ~~ошибку~~ сумму квадратичных ошибок. Она является величиной, которую можно минимизировать с помощью стандартной процедуры ~~ошибки~~ линейной регрессии. Теперь можно понять, с чем это связано: минимизация суммы квадратичных ошибок позволяет получить линейную модель с максимальным правдоподобием, при условии, что данные вырабатывались с гауссовым шумом, имеющим постоянную дисперсию.

## Обучение байесовским параметрам

Метод обучения с максимальным правдоподобием может стать основой некоторых очень простых процедур, но обнаруживает определенные серьезные недостатки при работе с небольшими наборами данных. Например, после обнаружения одного вишневого леденца в этом методе вырабатывается гипотеза с максимальным правдоподобием, что данный пакет на 100% состоит из вишневых леденцов (т.е.  $\theta = 1.0$ ). Но если только принятое распределение априорных вероятностей гипотезы не сводится к тому, что в пакетах должны находиться лишь одни вишневые леденцы либо исключительно лимонные леденцы, то такое заключение является необоснованным. В байесовском подходе к обучению параметрам распределению априорных вероят-

ностей гипотезы дается предпочтение над возможными значениями параметров, а само распределение обновляется по мере поступления данных.

В примере с конфетами, приведенном на рис. 20.2, *a*, имеется только один параметр,  $\theta$ , — вероятность того, что случайно выбранная конфета относится к разновидности вишневых леденцов. С точки зрения байесовского подхода  $\theta$  представляет собой (неизвестное) значение случайной переменной  $\Theta$ ; распределение априорных вероятностей гипотезы представляет собой распределение априорных вероятностей  $P(\Theta)$ . Таким образом,  $P(\Theta=\theta)$  — это априорная вероятность того, что в пакете имеется доля  $\theta$  вишневых леденцов.

Если параметр  $\theta$  может иметь любое значение от 0 до 1, то  $P(\Theta)$  должно представлять собой непрерывное распределение, которое является ненулевым только между 0 и 1 и интеграл которого равен 1. Одним из потенциальных распределений, пригодных для этой роли, является распределение с равномерной плотностью  $P(\Theta)=U[0,1](\theta)$  (см. главу 13). Как оказалось, распределение с равномерной плотностью является членом семейства **бета-распределений**. Каждое бета-распределение определяется двумя **гиперпараметрами**<sup>4</sup>, *a* и *b*, такими, что справедливо следующее соотношение для  $\theta$  в диапазоне значений  $[0, 1]$ :

$$\text{beta } [a, b](\theta) = \alpha \theta^{a-1} (1-\theta)^{b-1} \quad (20.6)$$

Константа нормализации  $\alpha$  зависит от *a* и *b* (см. упр. 20.8). На рис. 20.5 показано, как выглядит это распределение при различных значениях *a* и *b*. Среднее значение этого распределения равно  $a / (a+b)$ , поэтому большие значения *a* показывают обоснованность убеждения, что  $\Theta$  ближе к 1, чем к 0. При больших значениях *a+b* распределение становится более заостренным, что выражает большую уверенность в правильности значения  $\Theta$ . Таким образом, семейство бета-распределений предоставляет удобный ряд возможностей выбора распределений априорных вероятностей гипотезы.

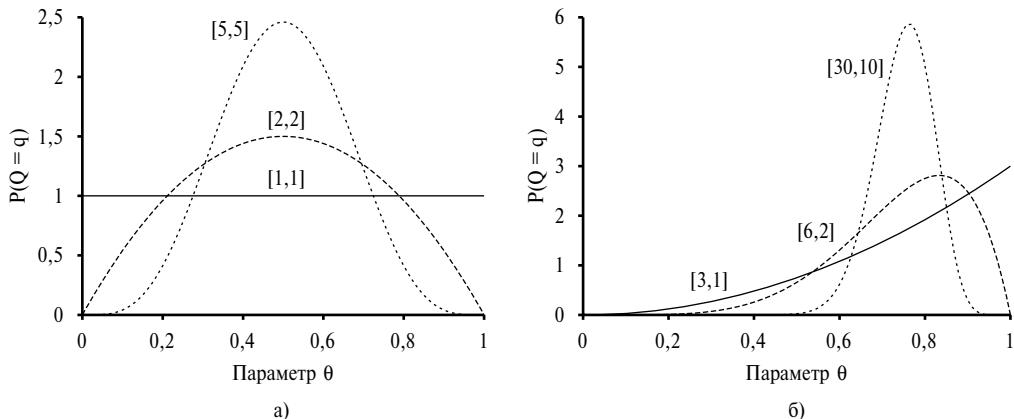


Рис. 20.5. Примеры распределения  $\text{beta } [a, b]$  для различных значений  $[a, b]$

<sup>4</sup> Они называются гиперпараметрами, поскольку параметризуют распределение по  $\theta$ , который сам является параметром.

Кроме такой гибкости, семейство бета-распределений обладает еще одним замечательным свойством: если переменная  $\Theta$  имеет распределение априорных вероятностей  $\text{beta}[a, b]$ , то после наблюдения в любой точке данных распределение апостериорных вероятностей для  $\Theta$  также становится бета-распределением. Семейство бета-распределений называется **сопряженным распределением априорных вероятностей** для семейства распределений, относящихся к некоторой булевой переменной<sup>5</sup>. Рассмотрим, как применяется это свойство. Предположим, что наблюдается появление вишневого леденца; в таком случае имеет место следующее соотношение:

$$\begin{aligned} P(\theta | D_1=\text{cherry}) &= \alpha P(D_1=\text{cherry} | \theta) P(\theta) \\ &= \alpha \cdot \theta \cdot \text{beta}[a, b](\theta) = \alpha \cdot \theta \cdot \theta^{a-1} (1-\theta)^{b-1} \\ &= \alpha \cdot \theta^a (1-\theta)^{b-1} = \text{beta}[a+1, b](\theta) \end{aligned}$$

Таким образом, после обнаружения вишневого леденца наращивается параметр  $a$  для получения нового распределения апостериорных вероятностей; аналогичным образом, после обнаружения лимонного леденца наращивается параметр  $b$ . Поэтому гиперпараметры  $a$  и  $b$  можно рассматривать как **виртуальные счетчики**, в том смысле, что распределение априорных вероятностей  $\text{beta}[a, b]$  ведет себя точно так же, как если бы обучение начиналось с равномерного распределения априорных вероятностей  $\text{beta}[1, 1]$ , после чего было фактически обнаружено  $a-1$  вишневых леденцов и  $b-1$  лимонных.

Изучая последовательность бета-распределений, соответствующих возрастающим значениям  $a$  и  $b$ , и поддерживая постоянные пропорции, можно наглядно продемонстрировать, как изменяется распределение апостериорных вероятностей по параметру  $\Theta$  по мере поступления новых данных. Например, предположим, что пакет с конфетами в действительности содержит 75% вишневых леденцов. На рис. 20.5, б показана последовательность распределений  $\text{beta}[3, 1]$ ,  $\text{beta}[6, 2]$ ,  $\text{beta}[30, 10]$ . Очевидно, что эта последовательность сходится к узкому пику вокруг истинного значения  $\Theta$ . Поэтому при наличии больших наборов данных процесс байесовского обучения постепенно сходится (по меньшей мере в данном случае) и позволяет получить такие же результаты, как и обучение с учетом максимального правдоподобия.

Сеть, показанная на рис. 20.2, б, имеет три параметра,  $\theta$ ,  $\theta_1$ , и  $\theta_2$ , где  $\theta_1$  — вероятность наличия красной обертки на вишневом леденце, а  $\theta_2$  — вероятность наличия красной обертки на лимонном леденце. Распределение априорных вероятностей байесовской гипотезы должно охватывать все три параметра; это означает, что необходимо задать распределение  $P(\theta, \theta_1, \theta_2)$ . Обычно предполагается, что соблюдается свойство **независимости параметров**, как показано ниже.

$$P(\theta, \theta_1, \theta_2) = P(\theta) P(\theta_1) P(\theta_2)$$

Согласно этому предположению, каждый параметр может иметь свое собственное бета-распределение, которое обновляется отдельно по мере поступления данных.

После того как была сформулирована идея, что неизвестные параметры могут быть представлены случайными переменными, такими как  $\Theta$ , из нее можно вывести естественное заключение, что эти параметры можно включить в саму байесовскую

---

<sup>5</sup> К другим сопряженным распределениям априорных вероятностей относятся семейство распределений **Дирихле** для параметров дискретного многомерного распределения и семейство распределений **нормальных-Висхарта** для параметров гауссова распределения [109].

сеть. Для этого также потребуется сделать копии переменных, описывающих каждый экземпляр. Например, если проверены три леденца, то для их описания потребуются переменные  $Flavor_1$ ,  $Flavor_2$ ,  $Flavor_3$ , а также  $Wrapper_1$ ,  $Wrapper_2$ ,  $Wrapper_3$ . Параметрическая переменная  $\Theta$  определяет вероятность каждой переменной  $Flavor_i$ :

$$P(Flavor_i = \text{cherry} | \Theta = \theta) = \theta$$

Аналогичным образом, вероятности оберток зависят от  $\Theta_1$  и  $\Theta_2$ , например:

$$P(Wrapper_i = \text{red} | Flavor_i = \text{cherry}, \Theta_1 = \theta_1) = \theta_1$$

Теперь весь байесовский процесс обучения можно сформулировать как задачу вероятностного вывода в байесовской сети, имеющей соответствующую структуру (рис. 20.6). Предсказание, касающееся нового экземпляра примера, можно получить, добавляя к сети новые переменные экземпляра, с тем условием, что значения некоторых из них можно будет определять с помощью запросов. Такая формулировка процессов обучения и предсказания наглядно показывает, что для байесовского обучения не требуется задавать дополнительные “принципы обучения”. Кроме того, это означает, что в действительности существует лишь единственный алгоритм обучения, т.е. алгоритм вероятностного вывода для байесовских сетей.

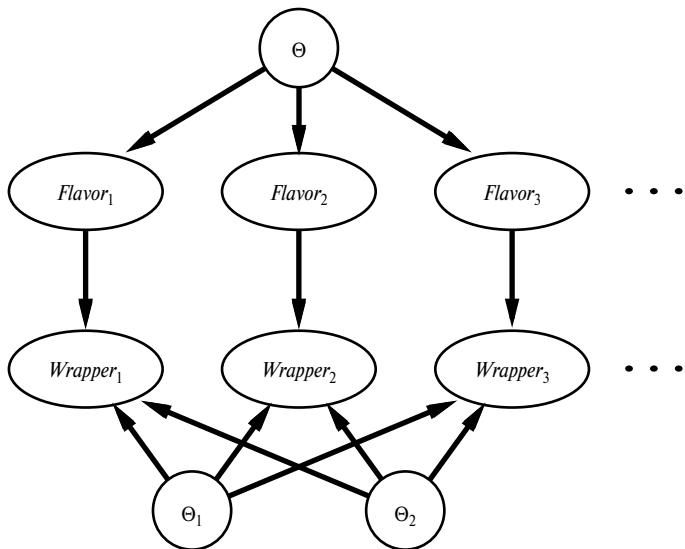


Рис. 20.6. Байесовская сеть, которая соответствует байесовскому процессу обучения. Распределения апостериорных вероятностей для параметрических переменных  $\Theta$ ,  $\Theta_1$  и  $\Theta_2$  можно определить путем вероятностного вывода на основании распределений апостериорных вероятностей этих параметрических переменных и свидетельств, касающихся переменных  $Wrapper_i$  и  $Flavor_i$

### Определение путем обучения структур байесовских сетей

До сих пор предполагалось, что структура байесовской сети задана, и мы просто пытаемся определить в процессе обучения ее параметры, тогда как структура самой

сети представляет основные причинные знания о проблемной области, которые часто может без особых затруднений сформулировать не только специалист, но даже неопытный пользователь. Но в некоторых случаях причинная модель может оказаться недоступной или стать предметом спора (например, некоторые корпорации долгое время утверждали, что курение не является причиной рака), поэтому важно понять, как может быть определена путем обучения структура байесовской сети на основе данных. В настоящее время алгоритмы структурного обучения находятся на начальном этапе развития, поэтому в данном разделе будет приведен лишь краткий обзор основных идей.

Наиболее очевидным подходом к решению этой задачи является поиск качественной модели. Эту работу можно начать с модели, не содержащей связей, и приступить к введению родительских узлов для каждого узла, согласуя параметры с помощью только что описанных методов и измеряя точность результирующей модели. Еще один вариант состоит в том, что можно начать с исходного предположения о структуре и использовать поиск с восхождением к вершине или с эмуляцией отжига для внесения модификаций, возвращая параметры после каждого изменения в структуре. Модификации могут включать обращение, добавление или удаление дуг. В этом процессе следует избегать появления циклов, поскольку во многих алгоритмах принято предположение, что для переменных задано упорядочение и что узел может иметь родительские узлы только среди тех узлов, которые присутствуют перед ним в этом упорядочении (точно так же, как и в процессе создания сети, описанном в главе 14). Для достижения полной общности необходимо также обеспечить поиск среди возможных упорядочений.

Существуют два альтернативных метода принятия решения о том, нужно ли прекратить поиск, поскольку обнаружена приемлемая структура. Первый из них предусматривает проверку того, действительно ли в данных удовлетворяются предположения об условной независимости, неявно заданные в этой структуре. Например, сам факт использования наивной байесовской модели для задачи с рестораном равносителен предположению о том, что справедливо приведенное ниже соотношение, поэтому можно проверить по самим данным, соблюдается ли это соотношение применительно к соответствующим условным частотам.

$$P(Fri/Sat, Bar | WillWait) = P(Fri/Sat | WillWait) \cdot P(Bar | WillWait)$$

Итак, даже если полученная структура описывает истинный причинный характер проблемной области, наличие статистических флюктуаций в наборе данных означает, что это уравнение никогда не будет выполняться точно, поэтому необходимо выполнить подходящую статистическую проверку для определения того, есть ли достаточно весомые свидетельства нарушения гипотезы о независимости. Сложность результирующей сети будет зависеть от пороговых значений, используемых для этой проверки — чем строже проверка на независимость, тем больше связей будет введено в сеть и тем выше опасность чрезмерно тщательной подгонки.

Подход, более совместимый с идеями, изложенными в этой главе, состоит в определении того, до какой степени предложенная модель объясняет данные (в вероятностном смысле). Но необходимо соблюдать осторожность при выборе способа измерения этой степени. Если будет просто предпринята попытка найти гипотезу с максимальным правдоподобием, то в конечном итоге будет получена полносвязная сеть, поскольку введение дополнительных родительских узлов для некоторого узла

не позволяет повысить его правдоподобие (см. упр. 20.9). Поэтому приходится каким-то образом вводить штраф за сложность модели. В подходе MAP (или MDL) просто вычитаются штрафные оценки из значений правдоподобия каждой структуры (после настройки параметров) до сравнения различных структур, а в байесовском подходе налагается совместное распределение априорных вероятностей на структуры и параметры. Но обычно количество структур, по которым должно быть выполнено суммирование, слишком велико (оно определяется суперэкспоненциальной зависимостью от количества переменных), поэтому большинство практиков используют алгоритм МСМС для формирования выборок по структурам.

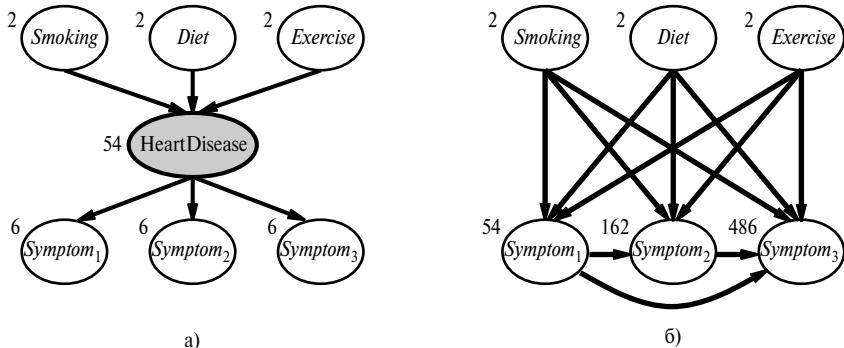
Применение способа штрафования за сложность (с помощью либо методов MAP, либо байесовских методов) влечет за собой появление важной связи между оптимальной структурой и характером представления для распределений условных вероятностей в сети. При использовании табличных распределений значения штрафов за сложность для распределения вероятностей узла растут экспоненциально в зависимости от количества родительских узлов, а при использовании, скажем, распределений зашумленного ИЛИ они растут только линейно. Это означает, что обучение с помощью моделей зашумленного ИЛИ (или других компактно параметризованных моделей), как правило, приводит к созданию в результате обучения таких структур, в которых имеется больше родительских узлов, чем при обучении с помощью табличных распределений.

### 20.3. ОБУЧЕНИЕ С ПОМОЩЬЮ СКРЫТЫХ ПЕРЕМЕННЫХ: АЛГОРИТМ ЕМ

Предыдущий раздел относился к полностью наблюдаемому случаю. Но многие реальные задачи характеризуются наличием **скрытых переменных** (иногда называемых ~~и~~ латентными переменными), которые не наблюдаются в данных, доступных для обучения. Например, истории болезни часто включают описания наблюдаемых симптомов, применяемого лечения и, возможно, результата лечения, но редко содержат сведения<sup>6</sup> о непосредственном наблюдении за развитием самого заболевания! Очевидно, напрашивается вопрос: “Если ход развития заболевания не наблюдается, то почему бы не построить модель без него?” Ответ на этот вопрос можно найти на рис. 20.7, где показана небольшая фиктивная диагностическая модель для сердечного заболевания. Существуют три наблюдаемых фактора, способствующих или препятствующих развитию заболевания, и три наблюдаемых симптома (которые имеют слишком сложные медицинские названия, поэтому им присвоены условные обозначения). Предполагается, что каждая переменная имеет три возможных значения (например, *none* (отсутствующий), *moderate* (умеренный) и *severe* (серезный)). Удаление скрытой переменной из сети, приведенной на рис. 20.7, *a*, влечет за собой создание сети, показанной на рис. 20.7, *b*; при этом общее количество параметров увеличивается с 78 до 708. Таким образом, ~~и~~ скрытые переменные

<sup>6</sup> В некоторых историях болезни содержатся сведения о диагнозе, предложенном лечащим врачом, но такой диагноз является причинным следствием симптомов, причиной которых, в свою очередь, стало заболевание.

позволяют резко уменьшить количество параметров, требуемых для определения байесовской сети. А такой факт, в свою очередь, позволяет резко уменьшить объем данных, необходимых для определения в процессе обучения рассматриваемых параметров.



*Рис. 20.7. Иллюстрация необходимости использования скрытых переменных: простая диагностическая сеть для сердечного заболевания, в которой предполагается наличие скрытой переменной; каждая переменная имеет три возможных значения и соответствующий ей узел обозначен количеством независимых параметров в распределении ее условной вероятности; общее количество параметров равно 78 (а); эквивалентная сеть с удаленным узлом HeartDisease. Обратите внимание на то, что переменные симптомов больше не являются условно независимыми, если даны значения их родительских переменных. Для этой сети требуется 708 параметров (б)*

Скрытые переменные важны, но их введение приводит к усложнению задачи обучения. Например, на рис. 20.7, *a* не показано наглядно, как найти в процессе обучения распределение условных вероятностей для переменной *HeartDisease*, если заданы ее родительские переменные, поскольку значение переменной *HeartDisease* в каждом случае не известно; такая же проблема возникает при поиске в процессе обучения распределений вероятностей для симптомов. В этом разделе описан алгоритм, называемый **ожиданием-максимизацией**, или сокращенно EM (Expectation-Maximization), который позволяет решить эту проблему очень общим способом. Мы рассмотрим три примера, а затем приведем общее описание. На первых порах складывается впечатление, что этот алгоритм действует как по волшебству, но как только будет достигнуто его интуитивное понимание, читатель сможет найти приложения для алгоритма EM в огромном спектре задач обучения.

## Неконтролируемая кластеризация: определение в процессе обучения смешанных гауссовых распределений

❖ **Неконтролируемой кластеризацией** называется задача выделения многочисленных категорий в коллекции объектов. Эта проблема — неконтролируемая, поскольку категориям не назначены метки. Например, предположим, что регистрируются спектры сотен тысяч звезд; звезды подразделяются на типы, которые можно определить по спектру, а если так оно и есть, то сколько таких типов и каковы их характеристики? Мы все знакомы с терминами наподобие “красный гигант” и “белый карлик”, но звезды не носят эти надписи на своих шляпах, поэтому астрономы должны

выполнять неконтролируемую кластеризацию для выявления их категорий. К другим примерам относится выявление видов, родов, отрядов и других категорий в таксономии живых организмов, установленной Линнеем, а также создание естественных разновидностей для распределения по категориям обычных объектов (см. главу 10).

Неконтролируемая кластеризация начинается с данных. На рис. 20.8, *a* показано 500 точек данных, каждая из которых задает значения двух непрерывных атрибутов. Точки данных могут соответствовать звездам, а атрибуты — интенсивностям спектра на двух определенных частотах. Затем необходимо понять, какого рода распределение вероятностей могло быть сформировано этими данными. Сама возможность кластеризации равносильна предположению, что данные сформированы с помощью смешанного распределения  $P$ . Такое распределение имеет  $k$  компонентов, каждый из которых представляет собой отдельное распределение. Точка данных формируется путем первоначального выбора компонента, а затем формирования выборки из этого компонента. Допустим, что этот компонент определяет случайная переменная  $C$  со значениями  $1, \dots, k$ ; в таком случае смешанное распределение задается следующим выражением:

$$P(\mathbf{x}) = \sum_{i=1}^k P(C=i) P(\mathbf{x}|C=i)$$

где  $\mathbf{x}$  относится к значениям атрибутов для точки данных. В случае непрерывных данных естественным вариантом выбора для распределений компонентов является многомерное гауссово распределение, что приводит к созданию семейства распределений, представляющего собой так называемое смешанное гауссово распределение. Параметрами смешанного гауссова распределения являются  $w_i = P(C=i)$  (вес каждого компонента),  $\mu_i$  (математическое ожидание каждого компонента) и  $\Sigma_i$  (ковариация каждого компонента). На рис. 20.8, *б* показано смешанное гауссово распределение, состоящее из трех распределений; фактически это смешанное распределение было источником данных, приведенных на рис. 20.8, *а*.

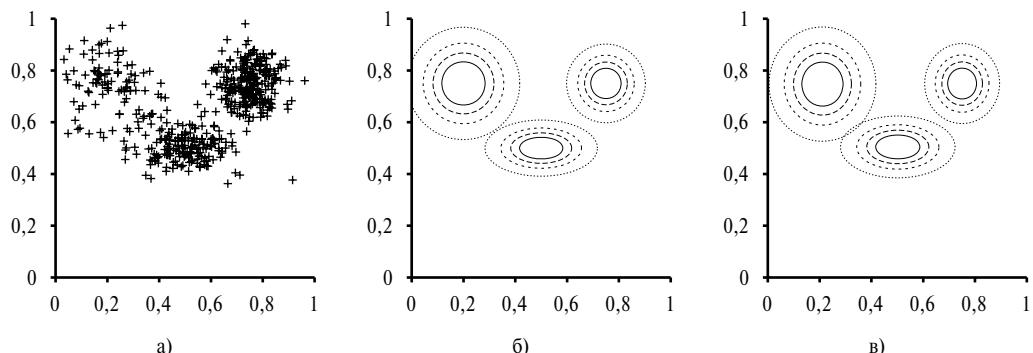


Рис. 20.8. Смешанное гауссово распределение: 500 двухмерных точек данных, показывающих наличие трех кластеров (*а*); модель смешанного гауссова распределения с тремя компонентами; веса компонентов (слева направо) равны 0.2, 0.3 и 0.5. Данные, приведенные на рис. 20.8, *а*, были сформированы с помощью этой модели (*б*); модель, реконструированная из данных, приведенных на рис. 20.8, *а*, с помощью алгоритма EM (*в*)

Таким образом, задача неконтролируемой кластеризации состоит в восстановлении модели смешанного распределения, подобной приведенной на рис. 20.8, б, из исходных данных, таких как показано на рис. 20.8, а. Очевидно, что если бы мы знали, с помощью какого компонента сформирована каждая точка данных, то было бы легко восстановить гауссово распределение для этого компонента, поскольку можно было просто выбрать все точки данных, соответствующие такому компоненту, а затем применить уравнение 20.4 (вернее, его многомерную версию) для согласования параметров гауссова распределения с множеством данных. С другой стороны, если бы были известны параметры каждого компонента, то можно было бы, по меньшей мере в вероятностном смысле, присвоить каждую точку данных компоненту. Но проблема состоит в том, что не известны ни присваивания, ни параметры.

Основная идея алгоритма EM, рассматриваемого в данном контексте, состоит в том, что необходимо принять допущение, будто нам известны параметры этой модели, а затем вычислить вероятность того, что каждая точка данных принадлежит к тому или иному компоненту. После этого нужно снова согласовать компоненты с данными таким образом, чтобы каждый компонент соглашался со всем набором данных, каждой точке которого назначен вес, соответствующий вероятности того, что она принадлежит к данному компоненту. Такой процесс продолжается итерационно до достижения сходимости. По сути при этом происходит “дополнение” данных путем вычисления распределений вероятностей по скрытым переменным (определяющим, какому компоненту принадлежит каждая точка данных) на основании текущей модели. При использовании смешанного гауссова распределения модель смешанного распределения инициализируется произвольными значениями параметров, а затем осуществляются итерации по описанным ниже двум шагам.

1. Е-шаг. Вычислить вероятности  $p_{ij} = P(C=i | \mathbf{x}_j)$  того, что данные  $\mathbf{x}_j$  были сформированы компонентом  $i$ . Согласно правилу Байеса, справедливо соотношение  $p_{ij} = \alpha P(\mathbf{x}_j | C=i) P(C=i)$ . Терм  $P(\mathbf{x}_j | C=i)$  представляет собой вероятность значений данных  $\mathbf{x}_j$  в  $i$ -м гауссовом распределении, а терм  $P(C=i)$  — это параметр определения веса  $i$ -го гауссова распределения; по определению  $p_i = \sum_j p_{ij}$ .
2. М-шаг. Вычислить новые значения математического ожидания, ковариаций и весов компонента следующим образом:

$$\begin{aligned}\mathbf{m}_i &\leftarrow \sum_j p_{ij} \mathbf{x}_j / p_i \\ \mathbf{s}_i &\leftarrow \sum_j p_{ij} \mathbf{x}_j \mathbf{x}_j^T / p_i \\ w_i &\leftarrow p_i\end{aligned}$$

Е-шаг, или шаг ожидания (expectation), может рассматриваться как вычисление ожидаемых значений  $p_{ij}$  скрытых ~~и~~ индикаторных переменных  $Z_{ij}$ , где значение  $Z_{ij}$  равно 1, если данные  $\mathbf{x}_j$  были сформированы  $i$ -м компонентом, и 0 — в противном случае. В М-шаге, или шаге максимизации (maximization), осуществляется поиск

новых значений параметров, которые максимизируют логарифмическое правдоподобие данных с учетом ожидаемых значений скрытых индикаторных переменных.

Окончательная модель, параметры которой определены в процессе обучения с помощью алгоритма EM, применяемого к данным, приведенным на рис. 20.8, *a*, показана на рис. 20.8, *в*; она практически не отличается от первоначальной модели, на основе которой были сформированы данные. На рис. 20.9, *a* показан график изменения логарифмического правдоподобия данных, соответствующих текущей модели, которое изменяется в процессе выполнения алгоритма EM. На этом графике заслуживают внимания две особенности. Во-первых, логарифмическое правдоподобие модели, окончательно полученной в процессе обучения, немного превышает соответствующее значение для первоначальной модели, на основании которой были сформированы исходные данные. Это явление может на первый взгляд показаться удивительным, но оно просто отражает тот факт, что данные были сформированы случайным образом, поэтому существует вероятность того, что они не являются точным отражением самой базовой модели. Во-вторых, любопытно то, что ~~в~~ в алгоритме EM логарифмическое правдоподобие данных повышается после каждой итерации. Можно доказать, что такова общая особенность данного алгоритма. Кроме того, можно доказать, что при определенных условиях алгоритм EM достигает локального максимума правдоподобия (а в редких случаях он может достичь точки перегиба или даже локального минимума). В этом смысле алгоритм EM напоминает алгоритм восхождения к вершине с учетом градиента, но заслуживает внимания то, что в нем уже не применяется параметр со “ступенчатым изменением величины”!

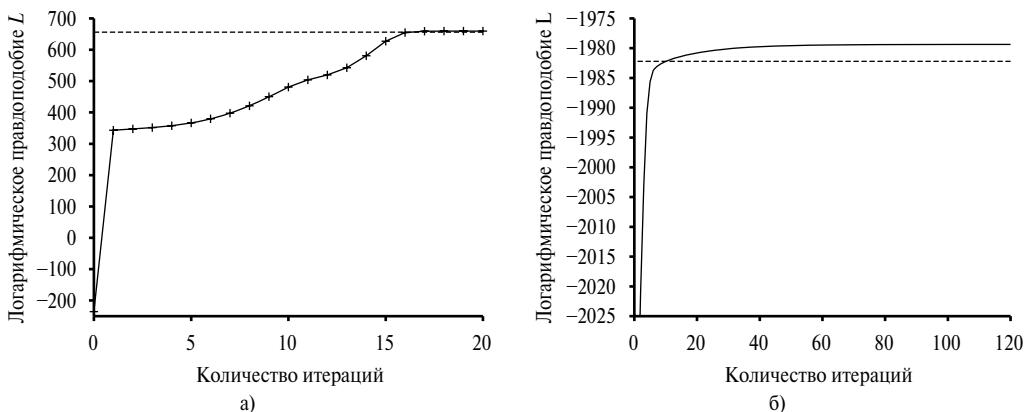


Рис. 20.9. Графики, показывающие изменение логарифмического правдоподобия данных,  $L$ , как функции от количества итераций EM. Горизонтальная линия соответствует логарифмическому правдоподобию истинной модели: график для модели смешанного гауссова распределения, показанной на рис. 20.8 (*а*); график для байесовской сети, приведенной на рис. 20.10, *а* (*б*)

Но события не всегда развиваются так удачно, как можно было бы судить на основании рис. 20.9, *а*. Например, может случиться так, что один компонент гауссова распределения сужится настолько, что будет охватывать лишь единственную точку данных. В таком случае его дисперсия достигнет нуля, а правдоподобие увеличится до бесконечности! Еще одна проблема состоит в том, что может произойти “слияние” двух компонентов, в результате чего они примут одинако-

вые значения средних и дисперсий, а точки данных станут для них общими. Вырожденные локальные максимумы такого рода представляют собой серьезную проблему, особенно в случае большого количества измерений. Одно из решений состоит в наложении распределений априорных вероятностей на параметры модели и применении версии МАР алгоритма ЕМ. Еще одно решение состоит в перезапуске вычислений для некоторого компонента с новыми случайно выбранными параметрами, если он становится слишком малым или слишком близким к другому компоненту. Иногда имеет также смысл выбирать для инициализации параметров более обоснованные значения.

### Обучение байесовских сетей со скрытыми переменными

Чтобы определить в процессе обучения параметры байесовской сети со скрытыми переменными, можно применить такие же подходы, которые позволили добиться успеха в случае смешанных гауссовых распределений. На рис. 20.10 показана ситуация, в которой имеются два пакета конфет, смешанных друг с другом. Для описания конфет применяются три характеристики: кроме разновидности *Flavor* и обертки *Wrapper*, в некоторых конфетах находятся леденцы с отверстиями *Hole* в середине, а в некоторых — леденцы без отверстий. Распределение конфет в каждом пакете описано с помощью **наивной байесовской** модели: в каждом конкретном пакете характеристики являются независимыми, но распределение условных вероятностей каждой характеристики зависит от пакета. Применяются следующие параметры:  $\theta$  — априорная вероятность того, что конфета взята из пакета *Bag 1*;  $\theta_{F1}$  и  $\theta_{F2}$  — вероятности того, что конфета относится к разновидности вишневых леденцов, при условии, что эта конфета взята из пакета *Bag 1* и *Bag 2* соответственно;  $\theta_{W1}$  и  $\theta_{W2}$  задают вероятности того, что обертка имеет красный цвет; а  $\theta_{H1}$  и  $\theta_{H2}$  — вероятности того, что леденец имеет отверстие. Обратите внимание на то, что вся эта модель в целом представляет собой модель смешанного распределения (в действительности это смешанное гауссово распределение можно также промоделировать в виде байесовской сети, как показано на рис. 20.10, б). На этом рисунке скрытая переменная соответствует пакету, поскольку после смешивания конфет мы больше не имеем возможности определить, из какого пакета взята каждая конфета. Можно ли в таком случае восстановить описание этих двух пакетов, наблюдая за характеристиками конфет, взятых из этой смеси?

Проведем одну итерацию алгоритма ЕМ для решения этой задачи. Вначале рассмотрим данные. Сформировано 1000 выборок из модели, истинными параметрами которой являются следующие:

$$\theta = 0.5, \theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8, \theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3 \quad (20.7)$$

Это означает, что равновероятно получение конфет из одного или другого пакета; в первом пакете в основном находятся вишневые леденцы в красных обертках и с отверстиями, а во втором — в основном лимонные леденцы в зеленых обертках и без отверстий. Количество восьми возможных разновидностей конфет определено в табл. 20.1.

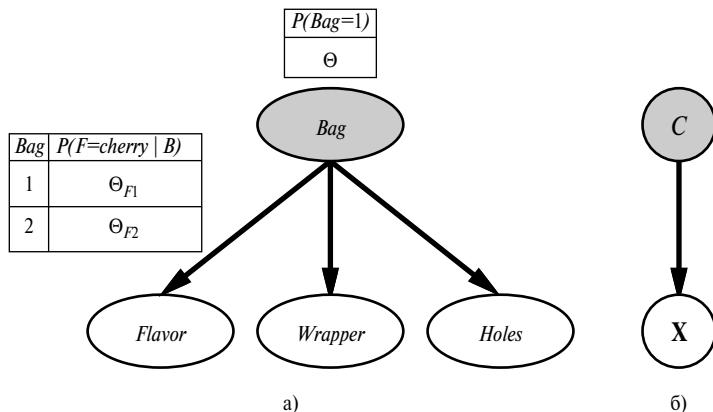


Рис. 20.10. Примеры применения модели смешанного гауссова распределения и байесовской сети: модель смешанного распределения для задачи с конфетами. Относительные количества различных разновидностей оберток и количества отверстий в леденцах зависят от пакета, который определен с помощью ненаблюдаемой переменной (а); байесовская сеть, соответствующая смешанному гауссову распределению. Среднее и ковариация наблюдаемой переменной  $\mathbf{x}$  зависит от компонента  $C$  (б)

Таблица 20.1. Восемь возможных разновидностей конфет

$W = \text{red}$		$W = \text{green}$	
$H = 1$	$H = 0$	$H = 1$	$H = 0$
$F = \text{cherry}$	273	93	104
$F = \text{lime}$	79	100	94
			167

Начнем с инициализации параметров. Для упрощения числовых расчетов выберем следующие значения параметров<sup>7</sup>:

$$\theta^{(0)} = 0.6, \quad \theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6, \quad \theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4 \quad (20.8)$$

Вначале проведем расчет для параметра  $\theta$ . В полностью наблюдаемом случае можно было бы получить оценку для этого параметра непосредственно из наблюдаемых значений количества конфет, относящихся к пакетам 1 и 2. Но поскольку номер пакета — это скрытая переменная, рассчитаем вместо этого ожидаемые значения количества. Ожидаемое количество  $\hat{N}(\text{Bag}=1)$  представляет собой сумму вероятностей того, что конфета взята из пакета 1, по всем конфетам:

$$\theta^{(1)} = \hat{N}(\text{Bag}=1) / N = \sum_{j=1}^N P(\text{Bag}=1 | flavor_j, wrapper_j, holes_j) / N$$

Эти вероятности можно вычислить с помощью любого алгоритма вероятностного вывода для байесовских сетей. А применительно к наивной байесовской модели, по-

<sup>7</sup> На практике лучше выбирать эти значения случайным образом для предотвращения появления локальных максимумов из-за симметрии.

доброй той, что рассматривается в данном примере, этот вероятностный вывод можно выполнить “вручную”, используя правило Байеса и применяя выражение для условной независимости:

$$\theta^{(1)} = \frac{1}{N} \sum_{j=1}^N \frac{P(\text{flavor}_j | \text{Bag}=1) P(\text{wrapper}_j | \text{Bag}=1) P(\text{holes}_j | \text{Bag}=1) P(\text{Bag}=1)}{\sum_i P(\text{flavor}_j | \text{Bag}=i) P(\text{wrapper}_j | \text{Bag}=i) P(\text{holes}_j | \text{Bag}=i) P(\text{Bag}=i)}$$

(Обратите внимание на то, что нормализующая константа также зависит от параметров.) Применяя эту формулу, например, к данным о 273 конфетах в красной обертке, среди которых находятся вишневые леденцы с отверстиями, определим, какой вклад они вносят в распределение вероятностей:

$$\frac{273}{1000} \cdot \frac{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)}}{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)} + \theta_{F2}^{(0)} \theta_{W2}^{(0)} \theta_{H2}^{(0)} (1-\theta^{(0)})} \approx 0.22797$$

Продолжая эти расчеты для семи других видов конфет, количество которых указано в табл. 20.1, получим, что  $\theta^{(1)} = 0.6124$ .

Теперь рассмотрим другие параметры, такие как  $\theta_{F1}$ . В полностью наблюдаемом случае это значение можно было бы оценить непосредственно на основе наблюдаемых значений количества вишневых и лимонных леденцов из пакета 1. Ожидаемое количество вишневых леденцов из пакета 1 задается с помощью следующего выражения:

$$\sum_{j: \text{Flavor}_j=\text{cherry}} P(\text{Bag}=1 | \text{Flavor}_j=\text{cherry}, \text{wrapper}_j, \text{holes}_j)$$

Эти вероятности также можно вычислить с помощью любого алгоритма для байесовской сети. Продолжая этот процесс, получим новые значения для всех параметров:

$$\begin{aligned} \theta^{(1)} &= 0.6124, \quad \theta_{F1}^{(1)} = 0.6684, \quad \theta_{W1}^{(1)} = 0.6483, \quad \theta_{H1}^{(1)} = 0.6558 \\ \theta_{F2}^{(1)} &= 0.3887, \quad \theta_{W2}^{(1)} = 0.3817, \quad \theta_{H2}^{(1)} = 0.3827 \end{aligned} \quad (20.9)$$

Логарифмическое правдоподобие данных возрастает от первоначального значения, примерно равного  $-2044$ , приблизительно до  $-2021$  после первой итерации, как показано на рис. 20.9, б. Это означает, что в данном обновлении само значение правдоподобия улучшается примерно на коэффициент  $e^{23} \approx 10^{10}$ . К десятой итерации модель, полученная в процессе обучения, лучше согласуется с данными, чем первоначальная модель ( $L=-1982.214$ ). Но дальнейший прогресс очень сильно замедляется. Такая ситуация в приложениях алгоритма EM встречается весьма часто, поэтому во многих практически применяемых системах алгоритм EM на последнем этапе обучения используется в сочетании с таким алгоритмом на основе градиента, как алгоритм Ньютона–Рафсона (см. главу 4).

Общий вывод, который может быть сделан на основании данного примера, состоит в том, что обновления параметров при обучении байесовской сети со скрытыми переменными являются непосредственно доступными из результатов вероятностного вывода по каждому примеру. Более того, для каждого параметра требуются только локальные априорные вероятности. Для общего случая, в котором в процессе обучения определяются параметры условной вероятности для каждой переменной  $X_i$ , если даны ее роди-

тельские переменные (иначе говоря,  $\theta_{ijk} = P(X_i=x_{ij} | Pa_i=pa_{ik})$ ), обновление задается с помощью нормализованных ожидаемых количеств следующим образом:

$$\hat{\theta}_{ijk} \leftarrow \hat{N}(X_i=x_{ij}, Pa_i=pa_{ik}) / \hat{N}(Pa_i=pa_{ik})$$

Эти ожидаемые количества можно получить путем суммирования по всем примерам и вычисления вероятностей  $P(X_i=x_{ij}, Pa_i=pa_{ik})$  для каждого из них с использованием любого алгоритма вероятностного вывода в байесовской сети. Для использования в точных алгоритмах (включая алгоритмы удаления переменных) все эти вероятности могут быть получены непосредственно как побочный продукт стандартного вероятностного вывода, без необходимости применения дополнительных вычислений, характерных для обучения. Более того, информация, необходимая для обучения, доступна локально применительно к каждому параметру.

## Обучение скрытых марковских моделей

Последнее приложение алгоритма EM, рассматриваемой в данной главе, касается изучения вероятностей перехода в скрытых марковских моделях (Hidden Markov Model — HMM). Напомним, что, как было сказано в главе 15, скрытая марковская модель может быть представлена с помощью динамической байесовской сети с одной дискретной переменной состояния (рис. 20.11). Каждая точка данных состоит из последовательности наблюдений конечной длины, поэтому задача заключается в том, чтобы определить в процессе обучения вероятности перехода на основании множества последовательностей наблюдений (или, возможно, на основании только одной длинной последовательности).

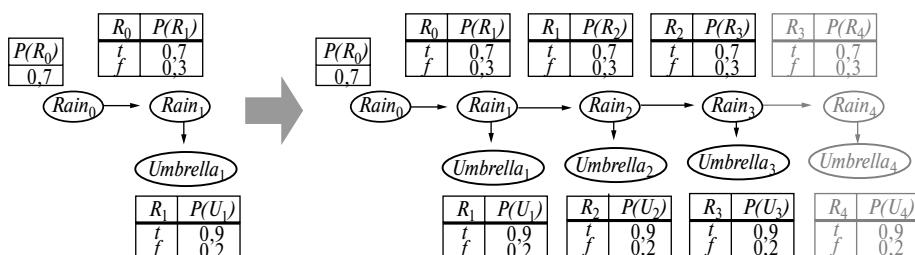


Рис. 20.11. Развернутая динамическая байесовская сеть, которая представляет скрытую марковскую модель (повторение рис. 15.12)

В данной главе уже было показано, как проводить обучение байесовских сетей, но в этом случае возникает одна сложность: в байесовских сетях каждый параметр является различным, а в скрытой марковской модели, с другой стороны, отдельные вероятности перехода из состояния  $i$  в состояния  $j$  во время  $t$ ,  $\theta_{ijt} = P(X_{t+1}=j | X_t=i)$ , повторяются во времени, т.е.  $\theta_{ijt}=\theta_{ij}$  для всех  $t$ . Чтобы оценить вероятность перехода из состояния  $i$  в состояние  $j$ , достаточно вычислить ожидаемую долю случаев, в которых система подвергается переходу в состояние  $j$ , находясь в состоянии  $i$ :

$$\hat{\theta}_{ij} \leftarrow \sum_t \hat{N}(X_{t+1}=j, X_t=i) / \sum_t \hat{N}(X_t=i)$$

Опять-таки эти ожидаемые количества могут быть вычислены с помощью любого алгоритма вероятностного вывода для скрытой марковской модели. Для вычисления необходимых вероятностей можно очень легко модифицировать **прямой-обратный** алгоритм, приведенный в листинге 15.1. Заслуживает внимание одно важное замечание, что требуются вероятности, полученные путем **сглаживания**, а не **фильтрации**; это означает, что при оценке вероятности того, что произошел конкретный переход, необходимо учитывать полученное впоследствии свидетельство. Например, как было указано в главе 15, свидетельство в случае убийства обычно может быть получено только после этого преступления (т.е. после перехода из состояния  $i$  в состояние  $j$ ).

## Общая форма алгоритма EM

Выше было описано несколько вариантов применения алгоритма EM. Каждый из них сводился к тому, что вычислялись ожидаемые значения скрытых переменных для каждого примера, а затем повторно вычислялись параметры с использованием ожидаемых значений так, как если бы они были наблюдаемыми значениями. Допустим, что  $\mathbf{x}$  — все наблюдаемые значения во всех примерах, и предположим, что  $\mathbf{z}$  обозначает все скрытые переменные для всех примеров, а  $\boldsymbol{\theta}$  представляет собой все параметры для модели вероятности. В таком случае алгоритм EM можно представить с помощью следующего уравнения:

$$\mathbf{q}^{(i+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\mathbf{z}} P(\mathbf{z}=\mathbf{z} | \mathbf{x}, \boldsymbol{\theta}^{(i)}) L(\mathbf{x}, \mathbf{z}=\mathbf{z} | \boldsymbol{\theta})$$

Это уравнение составляет саму суть алгоритма EM. При этом E-шаг сводится к вычислению выражения для суммы, которая представляет собой ожидаемое значение логарифмического правдоподобия “дополненных” данных по отношению к распределению  $P(\mathbf{z}=\mathbf{z} | \mathbf{x}, \boldsymbol{\theta}^{(i)})$ , т.е. распределение апостериорных вероятностей по скрытым переменным при наличии полученных данных. С другой стороны, M-шаг представляет собой максимизацию этого ожидаемого логарифмического правдоподобия по отношению к параметрам. При использовании смешанного гауссова распределения скрытыми переменными являются  $Z_{ij}$ , где  $Z_{ij}$  равна 1, если пример  $j$  был сформирован компонентом  $i$ . При использовании байесовских сетей скрытые переменные представляют собой значения ненаблюдаемых переменных для каждого примера. При использовании скрытых марковских моделей скрытые переменные являются переходами  $i \rightarrow j$ . Начиная с этой общей формы, можно вывести алгоритм EM для конкретного приложения, как только определены соответствующие скрытые переменные.

Поняв основную идею алгоритма EM, можно легко вывести все возможные его варианты и усовершенствования. Например, во многих случаях E-шаг (вычисление распределений апостериорных вероятностей по скрытым переменным) является трудновыполнимым, как при использовании больших байесовских сетей. Оказалось, что в этом случае можно применить приближенный E-шаг и все еще получить эффективный алгоритм обучения. А если используется такой алгоритм формирования выборки, как МСМС (см. раздел 14.5), то процесс обучения становится полностью интуитивно понятным: каждое состояние (конфигурация скрытых и наблюдаемых переменных), посещенное алгоритмом МСМС, рассматривается точно так

же, как если бы оно было полным наблюдением. Поэтому параметры могут обновляться непосредственно после каждого перехода из состояния в состояние в алгоритме МСМС. Для определения с помощью обучения параметров очень больших сетей показали свою эффективность и другие формы приближенного вероятностного вывода, такие как вариационные и циклические методы.

### Определение с помощью обучения структур байесовских сетей со скрытыми переменными

В разделе 20.2 обсуждалась задача определения с помощью обучения структур байесовских сетей на основе полных данных. А если приходится учитывать скрытые переменные, то задача усложняется. В простейшем случае скрытые переменные могут быть включены в общий список наряду с наблюдаемыми переменными; хотя их значения не наблюдаются, алгоритм обучения получает сведения о том, что они существуют, и должен найти для них место в структуре сети. Например, с помощью алгоритма может быть предпринята попытка определить в процессе обучения структуру, показанную на рис. 20.7, *а*, на основе той информации, что в модель должна быть включена переменная *HeartDisease* (трехзначная переменная). Если же алгоритму обучения не предоставлена эта информация, то в процессе обучения возникают два варианта: либо исходить из того, что данные действительно являются полными (что вынуждает алгоритм определить в процессе обучения модель с гораздо большим количеством параметров, показанную на рис. 20.7, *б*), либо изобрести новые скрытые переменные для упрощения модели. Последний подход можно реализовать путем включения новых вариантов операций модификации в процедуру поиска структуры — предусмотреть в алгоритме возможность не только модифицировать связи, но и добавлять или удалять скрытые переменные либо менять их арность. Безусловно, в таком алгоритме нельзя предусмотреть возможность назвать вновь изобретенную переменную, допустим *HeartDisease*, а также присваивать ее значениям осмыслиенные имена. Но, к счастью, вновь изобретенные скрытые переменные обычно связываются с ранее существовавшими переменными, поэтому человек — специалист в данной проблемной области часто может проверять локальные распределения условных вероятностей, касающиеся новой переменной, и устанавливать ее смысл.

Как и в случае полных данных, процесс определения структуры с помощью обучения с учетом максимального правдоподобия в чистом виде приводит к созданию полносвязной сети (более того, сети без скрытых переменных), поэтому требуется ввести какую-то форму штрафования за сложность. Для аппроксимации байесовского обучения можно также применить алгоритм МСМС. Например, можно предусмотреть определение в процессе обучения параметров смешанного гауссова распределения с неизвестным количеством компонентов, осуществляя выборки по такому количеству; приближенное распределение апостериорных вероятностей для количества заданных гауссовых распределений определяется частотами выборки в процессе применения алгоритма МСМС.

До сих пор рассматриваемый процесс обучения состоял из внешнего цикла, который представлял собой процесс поиска структуры, и внутреннего цикла, представляющего собой процесс оптимизации параметров. В случае полных данных внутренний цикл выполняется очень быстро, поскольку при этом достаточно лишь

извлечь информацию об условных частотах из набора данных. Если же имеются скрытые переменные, то для выполнения внутреннего цикла может потребоваться применить много итераций алгоритма EM или алгоритма на основе градиента, а каждая итерация потребует вычисления распределений апостериорных вероятностей в байесовской сети, что само представляет собой NP-трудную задачу. К настоящему времени доказано, что такой подход является практически не применимым для определения в процессе обучения сложных моделей. Одно из возможных усовершенствований состоит в использовании так называемого алгоритма **структурного EM**, который действует во многом таким же образом, как и обычный (параметрический) алгоритм EM, за исключением того, что этот алгоритм может обновлять не только параметры, но и структуру. Так же как в обычном алгоритме EM используются текущие параметры для вычисления ожидаемых количеств в E-шаге, а затем эти количества применяются в M-шаге для выбора новых параметров, так и в алгоритме структурного EM используется структура для вычисления ожидаемых количеств, после чего эти количества применяются в M-шаге для оценки правдоподобия потенциальных новых структур (в этом состоит отличие данного метода от метода внешнего цикла/внутреннего цикла, в котором вычисляются новые ожидаемые количества для каждой потенциальной структуры). Таким образом, структурный алгоритм EM позволяет вносить в сеть несколько структурных изменений без каких-либо повторных вычислений ожидаемых количеств и обладает способностью определять в процессе обучения нетривиальные структуры байесовских сетей. Тем не менее необходимо проделать еще очень много работы, прежде чем можно будет утверждать, что задача определения структуры в процессе обучения окончательно решена.

---

## 20.4. ОБУЧЕНИЕ НА ОСНОВЕ ЭКЗЕМПЛЯРА

---

До сих пор приведенное в данной главе описание статистического обучения сосредоточивалось в основном на задаче подгонки параметров ограниченного семейства вероятностных моделей к неограниченному набору данных. Например, в методе неконтролируемой кластеризации используются смешанные гауссовые распределения на основании того предположения, что структуру рассматриваемых данных можно объяснить, трактуя их как сумму постоянного количества гауссовых распределений. Авторы настоящей книги называют такие методы **параметрическим обучением**. Методы параметрического обучения часто бывают простыми и эффективными, но предположение о том, что в данных воплощено конкретное ограниченное семейство моделей, часто слишком упрощает то, что происходит в реальном мире, из которого поступают эти данные. Верно, что при наличии очень малого объема данных нельзя надеяться определить в процессе обучения параметры сложной и подробной модели, но представляется неразумным по-прежнему придерживаться гипотезы с той же фиксированной сложностью, даже после того, как доступный набор данных становится очень большим!

В отличие от параметрического обучения, методы **непараметрического обучения** позволяют увеличивать сложность гипотезы по мере роста объема данных. Чем больше данных поступает в распоряжение исследователя, тем более развитой может становиться гипотеза. В данном разделе рассматриваются два очень простых семейства методов непараметрического **обучения на основе экземпляра** (или **обучения на основе**

**содержимого памяти**), получивших такое название потому, что они позволяют конструировать гипотезы непосредственно на основе самих обучающих экземпляров.

## Модели ближайшего соседа

Ключевая идея моделей **ближайшего соседа** состоит в том, что свойства любой конкретной входной точки  $\mathbf{x}$ , по-видимому, должны быть подобными свойствам точек, соседних по отношению к  $\mathbf{x}$ . Например, если требуется выполнить **оценку плотности** (т.е. оценить значение неизвестной плотности вероятности в точке  $\mathbf{x}$ ), то можно просто измерить ту плотность, с какой расположены точки, рассеянные в окрестности  $\mathbf{x}$ . Такая задача на первый взгляд может показаться очень простой, пока не станет очевидно, что нужно точно определить, что подразумевается под понятием “окрестность”. Если окрестность слишком мала, то не будет содержать ни одной точки данных, а если слишком велика, то может включить все точки данных, в результате чего будет получена всюду одинаковая оценка плотности. Одно из возможных решений состоит в том, чтобы определить окрестность как достаточно большую для включения  $k$  точек, где  $k$  достаточно велико для обеспечения получения осмысленной оценки. При постоянном значении  $k$  размеры окрестности изменяются — если данные являются разреженными, то окрестность велика, а если данные расположены плотно, то окрестность мала. На рис. 20.12, *a* показан пример данных, распределенных в двух измерениях, а на рис. 20.13 приведены результаты оценки плотности по  $k$  ближайшим соседним точкам на основании этих данных при  $k=3, 10$  и  $40$  соответственно. При  $k=3$  оценка плотности в любой точке основана только на 3 соседних точках и весьма изменчива. При  $k=10$  полученная оценка представляет собой хорошую реконструкцию истинной плотности, показанной на рис. 20.12, *б*. При  $k=40$  окрестность становится слишком большой и информация о структуре данных полностью теряется. На практике хорошие результаты для большинства наборов данных с малым количеством размерностей можно получить с помощью значения  $k$ , находящегося примерно между 5 и 10. Подходящее значение для  $k$  можно также выбрать с использованием перекрестной проверки.

Для выявления соседних точек, ближайших к точке запроса, нужна метрика расстояний  $D(\mathbf{x}_1, \mathbf{x}_2)$ . В двухмерном примере, приведенном на рис. 20.12, используется евклидово расстояние. Но такая метрика становится неподходящей, если каждая размерность пространства измеряет что-то другое (например, рост и вес), поскольку изменение масштаба одной размерности приводит к изменению множества ближайших соседних точек. Одним из решений является стандартизация масштаба для каждой размерности. Для этого измеряется среднеквадратичное отклонение каждой характеристики по всему набору данных и значения характеристик выражаются как кратные среднеквадратичного отклонения для этой характеристики (это — частный случай **расстояния Махalanобиса**, в котором также учитывается ковариация характеристик). Наконец, для дискретных характеристик можно использовать **расстояние Хемминга**, в котором  $D(\mathbf{x}_1, \mathbf{x}_2)$  определяется как количество характеристик, по которым отличаются точки  $\mathbf{x}_1$  и  $\mathbf{x}_2$ .

Оценки плотности, подобные приведенным на рис. 20.13, определяют совместные распределения по пространству входных данных. Но в отличие от байесовской сети, представление на основе экземпляра не может содержать скрытых перемен-

ных, а это означает, что может выполняться неконтролируемая кластеризация, как это было в случае с моделью смешанного гауссова распределения. Тем не менее оценка плотности все еще может использоваться для предсказания целевого значения  $y$  по входным значениям характеристики  $\mathbf{x}$  путем вычисления вероятности  $P(y|\mathbf{x}) = P(y, \mathbf{x}) / P(\mathbf{x})$ , при условии, что обучающие данные включают значения для соответствующей целевой характеристики.

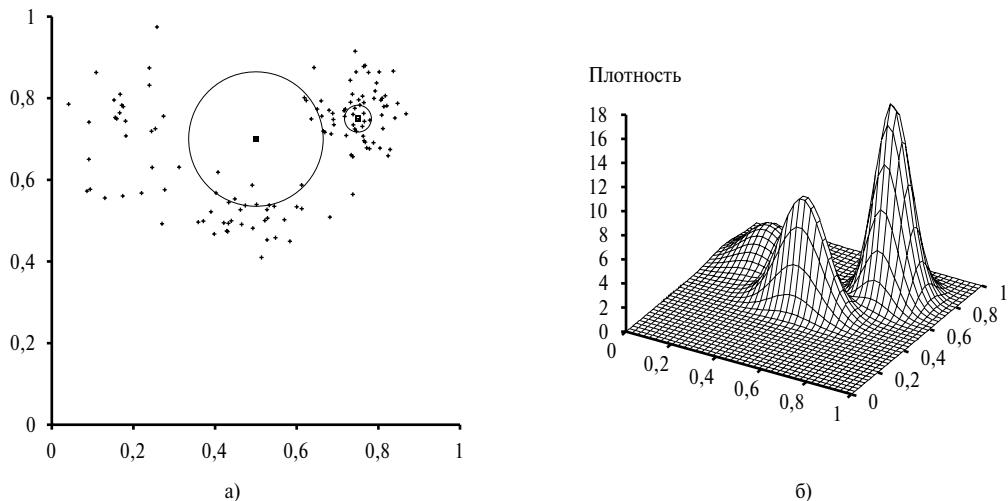


Рис. 20.12. Пример применения оценки плотности с  $k$  ближайшими соседними точками: частичная выборка данных, показанных на рис. 20.8, а, состоящая из 128 точек, наряду с двумя точками запроса и их окрестностями, которые включают 10 ближайших соседних точек (а); трехмерный график смешанного гауссова распределения, на основании которого были получены эти данные (б)

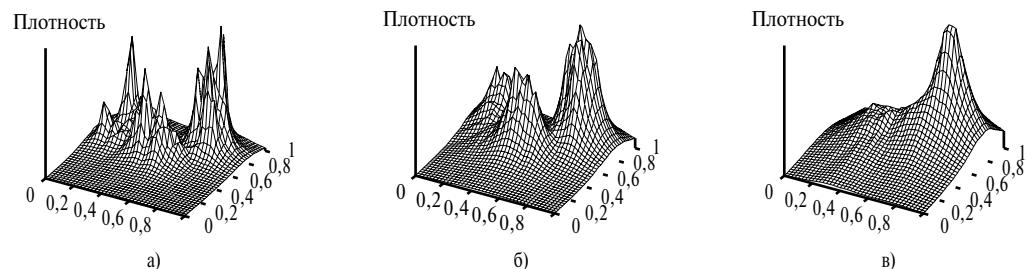


Рис. 20.13. Результаты применения метода оценки плотности с использованием окрестностей, включающих  $k$  ближайших соседних точек, к данным, приведенным на рис. 20.12, а, при  $k=3, 10$  и  $40$  соответственно

Идею оценки характеристик с помощью ближайшей соседней точки можно также непосредственно использовать в контролируемом обучении. Если имеется проверочный пример с входными данными  $\mathbf{x}$ , то выходное значение  $y=h(\mathbf{x})$  можно получить на основании значений  $y$  для  $k$  ближайших соседних точек точки  $\mathbf{x}$ . В дискретном случае единственное предсказание можно получить с помощью мажоритарного голосования, а в непрерывном случае — предусмотреть усреднение по  $k$  значениям

или применить локальную линейную регрессию, подгоняя гиперплоскость к  $k$  точкам и предсказывая значение в точке  $\mathbf{x}$  с помощью этой гиперплоскости.

Алгоритм обучения с помощью  $k$  ближайших соседних точек является очень простым для реализации, требует весьма небольшой настройки и часто показывает достаточно приемлемую производительность. Вполне целесообразно, столкнувшись с новой задачей обучения, вначале попытаться воспользоваться этим алгоритмом. Но при наличии больших наборов данных требуется эффективный механизм поиска соседних точек, ближайших к точке запроса  $\mathbf{x}$ , поскольку выполнение метода, в котором просто вычисляется расстояние до каждой точки, занимает слишком много времени. Было предложено много остроумных методов, позволяющих повысить эффективность этого этапа, которые основаны на предварительной обработке обучающих данных. Но, к сожалению, большинство из этих методов не достаточно хорошо масштабируются с увеличением количества размерностей пространства (т.е. количества характеристики).

При использовании многомерных пространств возникает еще одна проблема, связанная с тем, что ближайшие соседние точки в таких пространствах в действительности часто находятся очень далеко! Рассмотрим набор данных с размером  $N$  в  $d$ -мерном единичном гиперкубе и предположим, что нужно найти гиперкубическую окрестность с размером  $b$  и объемом  $b^d$  (те же рассуждения относятся к гиперсфера姆, но формула объема гиперсферы является более сложной). Чтобы в нее вошло  $k$  точек, средняя окрестность должна занимать долю  $k/N$  всего объема гиперкуба, который равен 1. Поэтому  $b^d = k/N$ , или  $b = (k/N)^{1/d}$ . До сих пор все шло хорошо. А теперь допустим, что количество характеристик  $d$  равно 100, и предположим, что  $k$  равно 10, а  $N$  равно 1 000 000. В таком случае получаем, что  $b \approx 0.89$ , т.е. что окрестность должна охватывать почти все пространство входных данных! Эти расчеты говорят о том, что методам с ближайшими соседними точками нельзя доверять, когда дело касается многомерных данных, а в случае малого количества размерностей не возникает никаких проблем; при  $d=2$  получаем  $b=0.003$ .

## Ядерные модели

В **ядерной модели** каждый обучающий экземпляр рассматривается как самостоятельно вырабатывающий небольшую функцию плотности (**ядерную функцию**). Оценка всей плотности в целом представляет собой нормализованную сумму всех небольших ядерных функций. Обучающий экземпляр в точке  $\mathbf{x}_i$  вырабатывает ядерную функцию  $K(\mathbf{x}, \mathbf{x}_i)$ , которая присваивает определенную вероятность каждой точке  $\mathbf{x}$  в пространстве. Поэтому оценка плотности принимает такой вид:

$$P(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N K(\mathbf{x}, \mathbf{x}_i)$$

Ядерная функция обычно зависит только от расстояния  $D(\mathbf{x}, \mathbf{x}_i)$  между точкой  $\mathbf{x}$  и экземпляром  $\mathbf{x}_i$ . Наиболее широко применяемой ядерной функцией (безусловно) является гауссово распределение. Для упрощения предполагается использование сферического гауссова распределения со среднеквадратичным отклонением  $w$  вдоль каждой оси, т.е. следующего распределения:

$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{(w^2 \sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x}, \mathbf{x}_i)^2}{2w^2}}$$

где  $d$  — количество размерностей в точке  $\mathbf{x}$ . И при таком подходе все еще приходится сталкиваться с проблемой выбора подходящего значения для  $w$ ; как и прежде, применение слишком малой окрестности приводит к получению оценки, состоящей из слишком большого количества пиков, как показано на рис. 20.14, *a*. На рис. 20.14, *b* видно, что промежуточное значение  $w$  позволяет получить очень хорошую реконструкцию первоначального распределения. А на рис. 20.14, *c* применение слишком большой окрестности приводит к полной потере структуры. Хорошее значение  $w$  может быть выбрано с использованием перекрестной проверки.

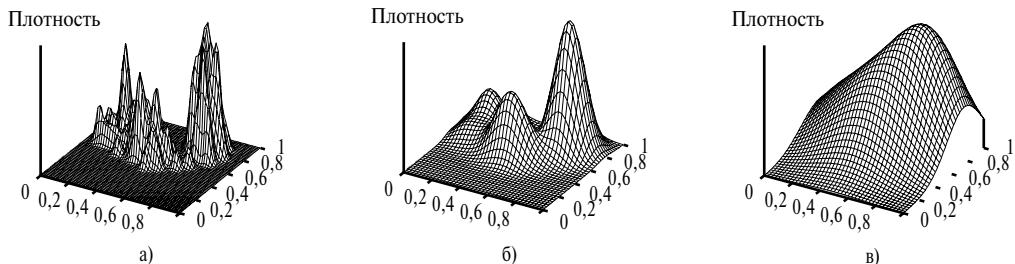


Рис. 20.14. Оценка плотности с помощью ядерной модели для данных, показанных на рис. 20.12, *a*, в которых используется ядерная функция гауссова распределения со значениями  $w=0.02, 0.07$  и  $0.20$

Контролируемое обучение с помощью ядерных функций осуществляется путем взятия взвешенных комбинаций всех предсказаний из обучающих экземпляров (сравните такой подход с предсказанием с помощью  $k$  ближайших соседних точек, в котором берется невзвешенная комбинация ближайших  $k$  экземпляров). Вес  $i$ -го экземпляра для точки запроса  $\mathbf{x}$  определяется по значению ядерной функции  $K(\mathbf{x}, \mathbf{x}_i)$ . Для предсказания в дискретном случае можно выполнить взвешенное голосование, а для предсказания в непрерывном случае — получить взвешенное среднее или применить метод взвешенной линейной регрессии. Следует отметить, что для получения предсказаний с помощью ядерной функции требуется анализ каждого обучающего экземпляра. Существует также возможность объединять ядерные функции со схемами индексации ближайших соседних точек для получения взвешенных предсказаний только с помощью самых близких экземпляров.

## 20.5. НЕЙРОННЫЕ СЕТИ

**Нейрон** — это клетка мозга или нервной системы, основной функцией которой является сбор, обработка и распространение электрических сигналов. Схематическое изображение типичного нейрона приведено на рис. 1.1 (см. с. 48). Считается, что способность мозга к обработке информации в основном обусловлена функционированием сетей, состоящих из таких нейронов. По этой причине целью некоторых из самых ранних работ по искусственному интеллекту было создание искусственных **нейронных сетей** (эта область научной деятельности упоминалась также под дру-

гими названиями, включая **коннекционизм**, **параллельная распределенная обработка** и **нейронные вычисления**). На рис. 20.15 показана простая математическая модель нейрона, предложенная Мак-Каллоком и Питтсом [1017]. Грубо говоря, нейрон “активизируется”, когда линейная комбинация значений на его входах превышает некоторый порог. Начиная с 1943 года были разработаны гораздо более подробные и реалистичные модели как для нейронов, так и для более крупных систем в мозгу, что привело к созданию такой современной научной области, как **вычислительная неврология**. С другой стороны, у исследователей в области искусственного интеллекта и статистики пробудился интерес к изучению более абстрактных свойств нейронных сетей, таких как способность выполнять распределенные вычисления, справляясь с зашумленными входными данными и обеспечивать обучение. Хотя со временем стало ясно, что подобные возможности предоставляют и другие системы (включая байесовские сети), нейронные сети остаются одной из наиболее широко применяемых и эффективных форм систем обучения и сами по себе могут стать важным предметом для изучения.

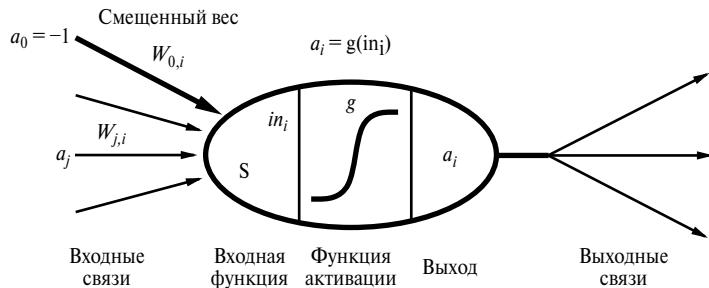


Рис. 20.15. Простая математическая модель нейрона. Выходной активацией этого элемента является

$$a_i = g \sum_{j=0}^n W_{j,i} a_j,$$

где  $a_j$  — выходная активация элемента  $j$ ;  $W_{j,i}$  — вес связи от элемента  $j$  к данному элементу

## Элементы в нейронных сетях

Нейронные сети состоят из узлов, или **элементов** (см. рис. 20.15), соединенных направленными **связями**. Связь от элемента  $j$  к элементу  $i$  служит для распространения **активации**  $a_j$  от  $j$  к  $i$ . Кроме того, каждая связь имеет назначенный ей числовой **вес**  $W_{j,i}$ , который определяет силу и знак связи. Каждый элемент  $i$  прежде всего вычисляет взвешенную сумму своих входных данных:

$$in_i = \sum_{j=0}^n W_{j,i} a_j$$

Затем он применяет к этой сумме **функцию активации**  $g$ , чтобы определить, какими должны быть выходные данные:

$$a_i = g(in_i) = g\left(\sum_{j=0}^n W_{j,i} a_j\right) \quad (20.10)$$

Обратите внимание на то, что в эту формулу входит **смещенный вес**  $W_{0,i}$ , относящийся к постоянному входному значению  $a_0 = -1$ . Роль, которую играет эта величина, будет описана немного позже.

Функция активации  $g$  предназначена для выполнения двух назначений. Во-первых, необходимо, чтобы элемент был “активным” (находился на уровне активации примерно +1) при наличии “правильных” входных данных и “неактивным” (с уровнем активации, близким к 0) при получении “неправильных” входных данных. Во-вторых, функция активации должна быть нелинейной, поскольку в противном случае произойдет сворачивание всех функций активации нейронной сети в простую линейную функцию (см. упр. 20.17). Два варианта формы функции активации  $g$  показаны на рис. 20.16 — **пороговая функция** и **сигмоидальная функция** (называемая также **логистической функцией**). Преимуществом сигмоидальной функции является то, что она дифференцируема, а это, как показано ниже, — важное свойство для алгоритма обучения с учетом весов. Обратите внимание на то, что обе функции имеют пороговое значение (либо жесткое, либо мягкое) около нуля; смещенный вес  $W_{0,i}$  задает фактическое пороговое значение для данного элемента в том смысле, что элемент активизируется после того, как взвешенная сумма “реальных” входных данных

$$\sum_{j=1}^n W_{j,i} a_j$$

(т.е. сумма, из которой исключен смещенный вход) превышает  $W_{0,i}$ .

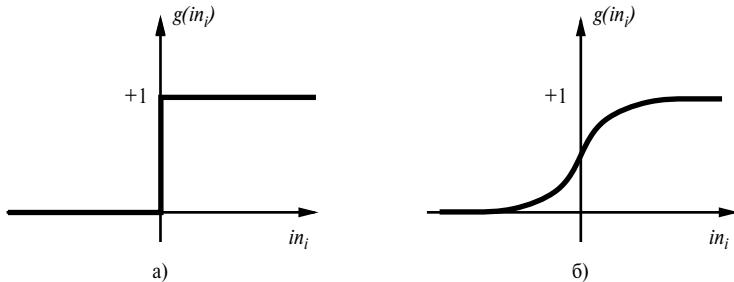


Рис. 20.16. Основные виды функций активации: **пороговая функция активации**, которая выводит 1, когда входные данные являются положительными, и 0 — в противном случае (иногда вместо нее используется **знаковая функция**, которая выводит  $\pm 1$  в зависимости от знака входного значения) (а); **сигмоидальная функция**  $1 / (1 + e^{-x})$  (б)

Представление о том, как работают отдельные элементы, можно получить, сравнив их с логическими элементами. Одной из первоначальных причин, по которым исследователи занялись проектированием отдельных элементов [1017], была способность элементов представлять основные булевые функции. На рис. 20.17 показано, как можно представить булевые функции AND, OR и NOT с помощью пороговых элементов, входам которых назначены подходящие веса. Такое свойство является

важным, поскольку оно означает, что эти элементы можно использовать для создания сети, обеспечивающей вычисление любой булевой функции от входных данных.

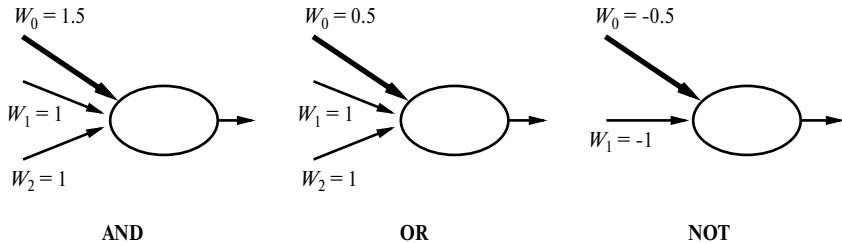


Рис. 20.17. Примеры элементов с пороговой функцией активации, которые могут действовать как логические элементы, если заданы соответствующие веса простых и смещенных входов

## Структуры сетей

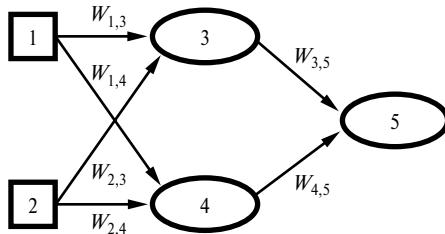
Существуют две основные категории структур нейронных сетей: ациклические сети, или **сети с прямым распространением**, и циклические, или **рекуррентные, сети**. Сеть с прямым распространением представляет определенную функцию ее текущих входных данных, поэтому не имеет внутреннего состояния, отличного от самих весов. Рекуррентная сеть, с другой стороны, подает свои выходные данные обратно на свои собственные входы. Это означает, что уровни активации сети образуют динамическую систему, которая может достигать устойчивого состояния, или переходить в колебательный режим, или даже проявлять хаотичное поведение. Более того, отклик сети на конкретные входные данные зависит от ее начального состояния, которое, в свою очередь, может зависеть от предыдущих входных данных. Поэтому рекуррентные сети (в отличие от сетей с прямым распространением) могут моделировать кратковременную память. Это означает, что они являются более интересными объектами для использования в качестве моделей мозга, но вместе с тем являются более трудными для понимания. В данном разделе в основном рассматриваются сети с прямым распространением; некоторые указания на источники для дополнительного чтения по рекуррентным сетям приведены в конце данной главы.

Проанализируем более внимательно утверждение о том, что сеть с прямым распространением представляет функцию от ее входных данных. Рассмотрим простую сеть, показанную на рис. 20.18, которая состоит из входных элементов, двух **скрытых элементов** и одного выходного элемента (чтобы упростить рассматриваемую схему, в данном примере удалены элементы, на которые подается смещение). Если задан вектор входных данных  $\mathbf{x} = (x_1, x_2)$ , активации входных элементов принимают вид  $(a_1, a_2) = (x_1, x_2)$ , а сеть вычисляет следующее значение:

$$\begin{aligned} a_5 &= g(W_{3,5}a_3 + W_{4,5}a_4) \\ &= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2)) \quad (20.11) \end{aligned}$$

Таким образом, выразив выходное значение каждого скрытого элемента как функцию его входных значений, мы показали, что выход всей сети,  $a_5$ , является функцией от ее входов. Кроме того, мы показали, что веса в сети действуют как параметры этой функции; если применить запись  $\mathbf{w}$  для обозначения параметров, то можно утверждать, что сеть вычисляет функцию  $h_{\mathbf{w}}(\mathbf{x})$ . Корректируя веса, можно

изменять функцию, представленную сетью. Именно так происходит обучение в нейронных сетях.



*Рис. 20.18. Очень простая нейронная сеть с двумя входными элементами, одним скрытым слоем из двух элементов и одного выходного элемента*

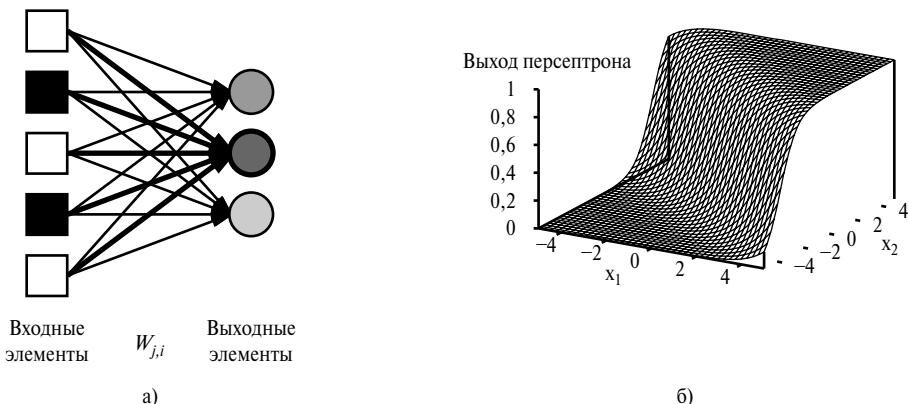
Нейронная сеть может использоваться для классификации или регрессии. Для булевой классификации с непрерывными выходными данными (например, формируемыми с помощью сигмоидальных элементов) обычно принято применять один выходной элемент, в котором значение, превышающее 0.5, интерпретируется как принадлежащее к одному классу, а значение ниже 0.5 — как принадлежащее к другому. Для  $k$ -сторонней классификации можно разделить диапазон одного выходного элемента на  $k$  частей, но чаще используется структура сети с  $k$  отдельными выходными элементами, притом что значение на каждом из них представляет относительное правдоподобие конкретного класса на основании текущих входных данных.

Сети с прямым распространением обычно размещаются по **слоям**, таким, что каждый элемент получает входные данные только от элементов, относящихся к непосредственно предшествующему слою. В следующих двух подразделах рассматриваются однослойные сети, не имеющие скрытых элементов, и многослойные сети, которые имеют один или несколько слоев скрытых элементов.

### Однослойные нейронные сети с прямым распространением (персептроны)

Сеть, в которой все входные элементы соединены непосредственно с выходными элементами, называется **однослойной нейронной сетью**, или сетью **персептрана**. Поскольку каждый выходной элемент является независимым от других (каждый вес влияет только на один из выходов), можно ограничиться в нашем исследовании рассмотрением персептронов с единственным выходным элементом, как показано на рис. 20.19, *a*.

Начнем с исследования пространства гипотез, которое может быть представлено с помощью персептрана. Если применяется пороговая функция активации, то персептрон может рассматриваться как представляющий некоторую булеву функцию. Кроме элементарных булевых функций AND, OR и NOT (см. рис. 20.17), персептрон позволяет представлять очень компактно некоторые весьма “сложные” булевые функции. Например, **мажоритарная функция**, которая выводит 1, только если 1 существует больше чем на половине из ее  $n$  входов, может быть представлена с помощью персептрана, в котором каждое значение  $W_j=1$ , а пороговое значение  $W_0=n/2$ . В дереве решений для представления этой функции потребовалось бы  $O(2^n)$  узлов.



*Рис. 20.19. Свойства сети персептрона: сеть персептрона, состоящая из трех выходных элементов персептрона, в которых совместно используются пять входных элементов (а). Рассматривая конкретный выходной элемент (скажем, второй элемент, выделенный жирным контуром), можно обнаружить, что веса его входных связей не влияют на другие выходные элементы; графическое изображение выходных данных элемента персептрона с двумя входами, имеющего сигмоидальную функцию активации (б)*

К сожалению, существует также много булевых функций, которые не могут быть представлены с помощью порогового персептрана. Рассматривая уравнение 20.10, можно обнаружить, что пороговый персептран возвращает 1 тогда и только тогда, когда взвешенная сумма его входных данных (включая смещение) является положительной:

$$\sum_{j=0}^n w_j x_j > 0 \text{ или } \mathbf{w} \cdot \mathbf{x} > 0$$

Итак,  $\mathbf{w} \cdot \mathbf{x} = 0$  определяет гиперплоскость в пространстве входных данных, поэтому персептрон возвращает 1 тогда и только тогда, когда входные данные находятся с одной стороны от этой гиперплоскости. По такой причине пороговый персептрон называют **линейным разделителем**. На рис. 20.20, *a*, *b* показана такая гиперплоскость (которая в двух измерениях является линией) для представления с помощью персептрана функций AND и OR от двух входных значений. Черные кружки обозначают точки в пространстве входных данных, где значение этой функции равно 1, а белые кружки — точки, где это значение равно 0. Персептрон способен представить эти функции, поскольку есть какая-то линия, которая отделяет все белые кружки от всех черных кружков. Такие функции называются **линейно разделимыми**. На рис. 20.20, *c* показан пример функции, которая не является линейно разделимой, — функции XOR. Безусловно, что пороговый персептрон не позволяет определить в процессе обучения эту функцию. Вообще говоря, *пороговые персептроны способны представить только линейно разделимые функции*. Но такие функции составляют лишь небольшую часть всех функций; в упр. 20.14 предлагается определить величину этой части. Сигмоидальные персептроны характеризуются аналогичными ограничениями, с той поправкой, что они представляют только “мягкие” линейные разделители (см. рис. 20.19, *b*).

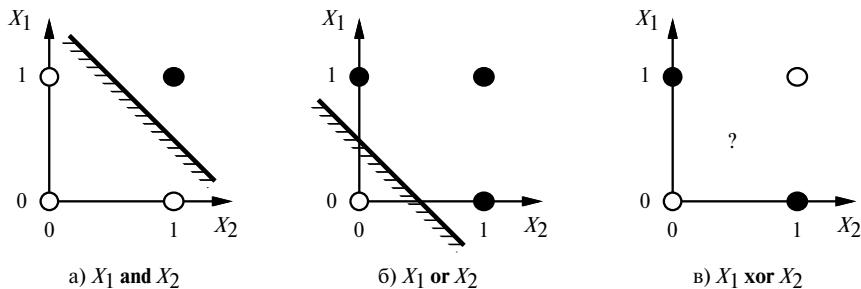


Рис. 20.20. Иллюстрация свойства линейной разделимости пороговых персептронов. Чёрные кружки показывают точки в пространстве входных данных, где значение функции равно 1, а белые кружки показывают точки, где это значение равно 0. Персептрон возвращает 1 при получении данных из области, соответствующей незаштрихованной стороне линии. На рис. 20.20, в не существует такой линии, которая могла бы правильно классифицировать входные данные

Несмотря на их ограниченную выразительную мощь, пороговые персептроны имеют некоторые преимущества. В частности, существует простой алгоритм обучения, позволяющий выполнить подгонку весов порогового персептрана к любому линейно разделимому обучающему множеству, но в данном разделе вместо описания этого алгоритма мы выведем тесно связанный с ним алгоритм для обучения с помощью сигмоидальных персептронов.

В основе этого алгоритма, а в действительности в основе большинства алгоритмов для обучения нейронных сетей, лежит такая идея, что в процессе обучения необходимо корректировать веса в сети для минимизации некоторого критерия измерения ошибок в обучающем множестве. Таким образом, задача обучения формулируется как некоторая задача оптимизационного поиска<sup>8</sup> в пространстве весов. “Классическим” критерием измерения ошибок является **сумма квадратичных ошибок**, которая использовалась в том числе в алгоритме линейной регрессии, приведенном на с. 956. Квадратичная ошибка для единственного обучающего примера с входными данными  $\mathbf{x}$  и истинными выходными данными  $y$  определяется следующим образом:

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

где  $h_{\mathbf{w}}(\mathbf{x})$  — выходные данные персептрана при обработке рассматриваемого примера.

Для уменьшения квадратичной ошибки можно применить метод градиентного спуска, вычисляя частичную производную  $E$  по отношению к каждому весу. При этом может быть получено следующее соотношение:

<sup>8</sup> Описание общих методов оптимизации, применимых к непрерывным пространствам, приведено в разделе 4.4.

$$\begin{aligned}
 \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} \\
 &= Err \times \frac{\partial}{\partial W_j} g\left(y - g\left(\sum_{j=0}^n W_j \cdot x_j\right)\right) \\
 &= -Err \times g'(in) \times x_j
 \end{aligned}$$

где  $g'$  — производная функции активации<sup>9</sup>. В алгоритме градиентного спуска предусмотрено, что если требуется уменьшить  $E$ , то вес обновляется следующим образом:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j \quad (20.12)$$

где  $\alpha$  — **скорость обучения**. Интуитивно ясно, что приведенное выше уравнение имеет большой смысл. Если ошибка  $Err=y-h_w(\mathbf{x})$  является положительной, то выход сети слишком мал и поэтому веса увеличиваются при положительных входных данных и уменьшаются при отрицательных входных данных. А если ошибка является отрицательной, то происходит противоположное<sup>10</sup>.

Полный алгоритм приведен в листинге 20.1. Он предусматривает прогон обучающих примеров через сеть каждый раз по одному и небольшую корректировку весов после прогона каждого примера для уменьшения ошибки. Каждый цикл прогона примеров называется **эпохой**. Эпохи повторяются до тех пор, пока не достигается некоторый критерий останова; как правило, такая ситуация достигается, когда изменения весов становятся очень небольшими. В других методах для всего обучающего набора вычисляется градиент путем сложения всех градиентных вкладов в уравнении 20.12 перед обновлением весов. А в методе **стохастического градиента** примеры выбираются случайным образом из обучающего набора вместо их циклической обработки.

**Листинг 20.1. Алгоритм обучения на основе градиентного спуска для персепtronов, в котором предполагается использование дифференцируемой функции активации  $g$ . Для пороговых персепtronов коэффициент  $g'(in)$  из обновления веса исключается. Функция Neural-Net-Hypothesis возвращает гипотезу, которая вычисляет выход сети для любого конкретного примера**

---

```

function Perceptron-Learning(examples, network) returns гипотеза
    персептрона
    inputs: examples, множество примеров, для каждого из которых заданы
        входные данные  $\mathbf{x}=x_1, \dots, x_n$  и выходные данные  $y$ 
        network, персептрон с весами  $W_j$ , где  $j=0 \dots n$ , и функцией
        активации  $g$ 

    repeat
        for each e in examples do
             $in \leftarrow \sum_{j=0}^n W_j \cdot x_j[e]$ 
             $Err \leftarrow y[e] - g(in)$ 

```

---

<sup>9</sup> Для сигмоидального персептрона эта производная задается выражением  $g' = g(1-g)$ .

<sup>10</sup> Для пороговых персепtronов, где производная  $g'(in)$  не определена, оригинальное **правило обучения персептрона**, разработанное Розенблattом [1302], идентично уравнению 20.12, однако в нем исключено значение  $g'(in)$ . Но поскольку значение  $g'(in)$  является одинаковым для всех весов, его исключение влияет только на величину, а не на направление общего обновления веса для каждого примера.

```

 $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$ 
until не достигается некоторый критерий останова
return Neural-Net-Hypothesis(network)

```

На рис. 20.21 показаны кривые обучения персептрона для двух разных задач. Слева показана кривая для определения в процессе обучения мажоритарной функции с 11 булевыми входами (т.е. функция выводит 1, если на шести или больше входах присутствует 1). Как и следовало ожидать, персептрон определяет эту функцию в процессе обучения очень быстро, поскольку мажоритарная функция является линейно разделимой. С другой стороны, обучающееся устройство, основанное на использовании дерева решений, не добивается существенного прогресса, поскольку мажоритарную функцию очень сложно (хотя и возможно) представить в виде дерева решений. Справа показана задача с ресторатором. Решение этой задачи можно легко представить в виде дерева решений, но соответствующая функция не является линейно разделимой. Наилучшая гиперплоскость, разделяющая данные, позволяет правильно классифицировать только 65% примеров.

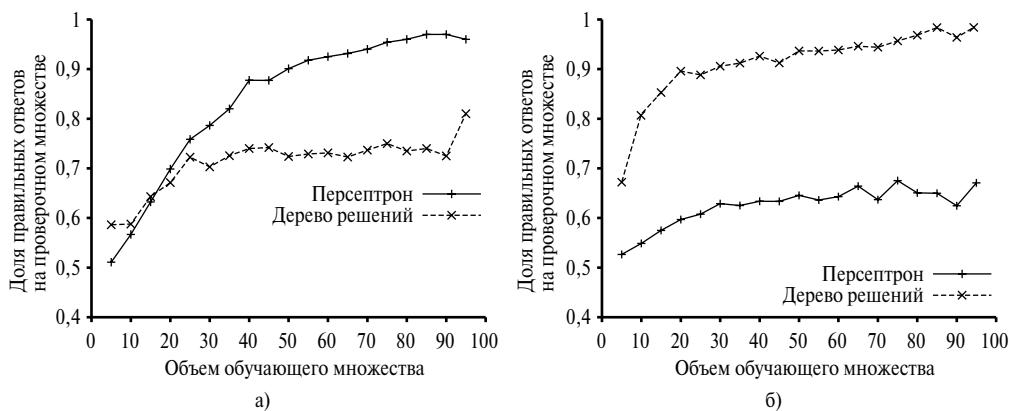


Рис. 20.21. Сравнение производительности персептронов и деревьев решений: персептроны показывают более высокую производительность при определении в процессе обучения мажоритарной функции от 11 входов (а); деревья решений показывают лучшую производительность при определении в процессе обучения предиката WillWait в задаче с ресторатором (б)

До сих пор персептроны рассматривались как средства реализации детерминированных функций, выходные данные которых могут содержать ошибки. Выход сигмоидального персептрона можно также интерпретировать как вероятность, а именно вероятность того, что истинным выходом является 1, если заданы все входы. При такой интерпретации сигмоидальный персептрон может использоваться как каноническое представление для условных распределений в байесовских сетях (см. раздел 14.3). Существует также возможность выполнять вероятностный вывод правила обучения с использованием стандартного метода максимизации (условного) логарифмического правдоподобия данных, как было описано выше в данной главе. Рассмотрим, как действует последний метод.

Допустим, что рассматривается единственный обучающий пример с истинным выходным значением  $T$ , и предположим, что  $p$  — вероятность, возвращаемая персептроном для этого примера. Если  $T=1$ , то условная вероятность этих данных равна

$p$ , а если  $T=0$ , то условная вероятность данных равна  $(1-p)$ . Теперь можно воспользоваться простым приемом, чтобы записать интересующее нас выражение для логарифмического правдоподобия в дифференцируемой форме. Этот прием состоит в том, что переменная со значениями 0/1 в экспоненте выражения рассматривается как **индикаторная переменная**:  $p^T - p$ , если  $T=1$ , и 1 — в противном случае; аналогичным образом,  $(1-p)^{(1-T)} - (1-p)$ , если  $T=0$ , и 1 — в противном случае. Поэтому можно записать выражение для логарифмического правдоподобия данных следующим образом:

$$L = \log p^T (1-p)^{(1-T)} = T \log p + (1-T) \log (1-p) \quad (20.13)$$

В силу самих свойств сигмоидальной функции полученный градиент сводится к очень простой формуле (упр. 20.16):

$$\frac{\partial L}{\partial W_j} = Err \times a_j$$

Обратите внимание на то, что ~~вектор обновления весов для обучения с максимальным правдоподобием в сигмоидальных персептронах по сути идентичен вектору обновления для минимизации квадратичной ошибки~~. Таким образом, можно утверждать, что персептроны имеют вероятностную интерпретацию, даже если правило их обучения выведено на основании детерминированного подхода.

### Многослойные нейронные сети с прямым распространением

Теперь рассмотрим сети со скрытыми элементами. Наиболее общий случай охватывает сети с одним скрытым слоем<sup>11</sup>, как показано на рис. 20.23. Преимущество введения скрытых слоев состоит в том, что они приводят к расширению пространства гипотез, которые могут быть представлены сетью. Каждый скрытый элемент можно рассматривать как персепtron, который представляет мягкую пороговую функцию в пространстве входов (см. рис. 20.19, б). В таком случае любой выходной элемент должен рассматриваться как средство создания линейной комбинации нескольких таких функций с мягким порогом. Например, путем сложения двух противоположно направленных мягких пороговых функций и определения порогового значения результата можно получить функцию с графиком в виде “хребта” (рис. 20.22, а). Соединение двух таких хребтов под прямыми углами друг к другу (например, комбинирование выходов четырех скрытых элементов) позволяет получить “выступ” (см. рис. 20.22, б).

<sup>11</sup> Некоторые специалисты называют сеть с одним скрытым слоем трехслойной сетью, а другие — двухслойной сетью (поскольку считают, что входные элементы не являются “настоящими” элементами). Чтобы избежать путаницы, авторы будут называть такую сеть “сетью с одним скрытым слоем”.

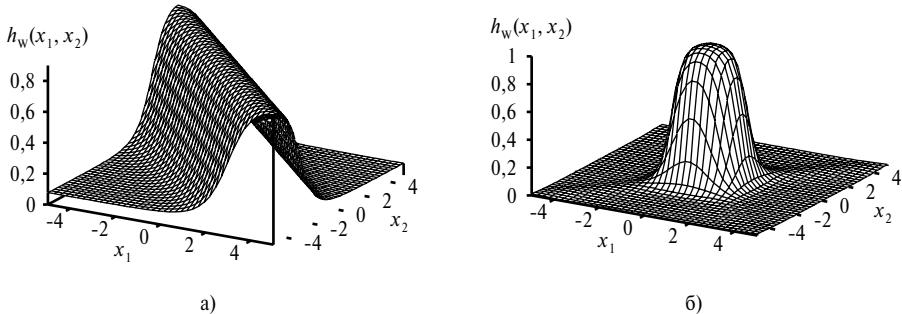


Рис. 20.22. Примеры применения мягких пороговых функций: результатом комбинирования двух противоположно направленных мягких пороговых функций для создания хребта (а); результатом комбинирования двух хребтов для создания выступа (б)

При наличии большего количества скрытых элементов появляется возможность создавать еще больше выступов различных размеров, расположенных в разных местах. В действительности с помощью одного, достаточно большого скрытого слоя возможно представить любую непрерывную функцию от входных данных с произвольной точностью, а при наличии двух слоев можно представить даже функции с разрывами<sup>12</sup>. К сожалению, применительно к любой конкретной сетевой структуре сложно точно охарактеризовать, какие функции могут быть представлены с ее помощью, а какие — нет.

Предположим, что необходимо сконструировать сеть со скрытым слоем для задачи с рестораном. Имеется 10 атрибутов, описывающих каждый пример, поэтому требуется 10 входных элементов. А сколько нужно скрытых элементов? На рис. 20.23 показана сеть с четырьмя скрытыми элементами. Как оказалось, такое количество скрытых элементов почти полностью подходит для решения данной задачи. Но проблема заблаговременного выбора подходящего количества скрытых элементов все еще полностью не исследована (с. 990).

Алгоритмы обучения для многослойных сетей подобны алгоритму обучения для персепtronов, приведенному в листинге 20.1. Одно небольшое различие состоит в том, что может быть предусмотрено несколько выходов, поэтому должен применяться вектор выходов  $\mathbf{h}_w(\mathbf{x})$ , а не одно значение, и с каждым примером должен быть связан вектор выходов  $\mathbf{y}$ . Между этими алгоритмами существует также важное различие, которое заключается в том, что ошибка  $\mathbf{y} - \mathbf{h}_w$  в выходном слое является очевидной, а ошибка в скрытых слоях кажется неуловимой, поскольку в обучающих данных отсутствует информация о том, какие значения должны иметь скрытые узлы. Как оказалось, можно обеспечить обратное распространение ошибки из выходного слоя в скрытые слои. Процесс обратного распространения может быть организован непосредственно на основе исследования общего градиента ошибки. Вначале мы опишем этот процесс на основе интуитивных представлений, а затем приведем его обоснование.

<sup>12</sup> Доказательство этого утверждения является сложным, но основное следствие из него состоит в том, что необходимое количество скрытых элементов растет экспоненциально в зависимости от количества входов. Например, чтобы закодировать все булевые функции от  $n$  входов, требуется  $2^n/n$  скрытых элементов.

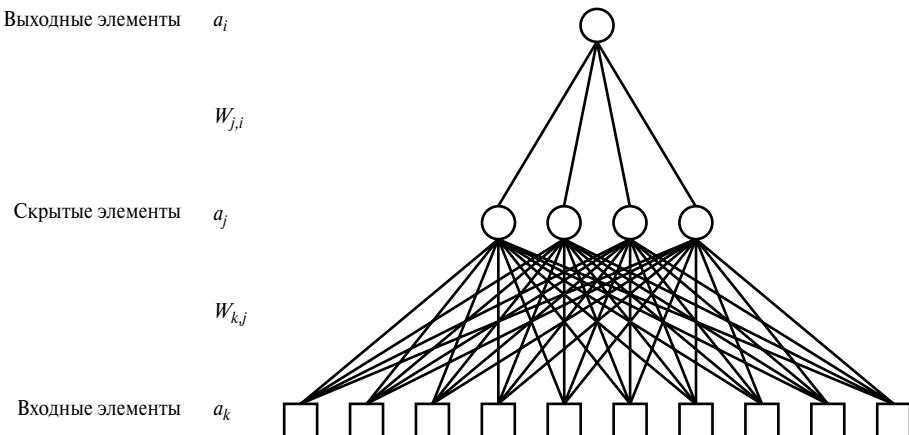


Рис. 20.23. Многослойная нейронная сеть с одним скрытым слоем и 10 входами, применимая для решения задачи с рестораном

Правило обновления весов, применяемое в выходном слое, идентично уравнению 20.12. Предусмотрено несколько выходных элементов, поэтому предположим, что  $Err_i$  является  $i$ -м компонентом вектора ошибки  $\mathbf{y}-\mathbf{h}_w$ . Авторы находят также удобным определить модифицированную ошибку  $\Delta_i = Err_i \times g'(in_i)$ , с помощью которой правило обновления весов можно преобразовать следующим образом:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad (20.14)$$

Чтобы обновить веса связей между входными и скрытыми элементами, необходимо определить величину, аналогичную терму ошибки для выходных узлов. Именно в этом и заключается суть метода обратного распространения ошибки. Идея его состоит в том, что скрытый узел  $j$  “отвечает” за некоторую долю ошибки  $\Delta_i$  в каждом из выходных узлов, с которыми он соединен. Таким образом, значения  $\Delta_i$  разделяются в соответствии с весом связи между скрытым узлом и выходным узлом и распространяются обратно, обеспечивая получение значений  $\Delta_j$  для скрытого слоя. Правило распространения для значений  $\Delta$  состоит в следующем:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \quad (20.15)$$

Теперь правило обновления весов между входными элементами и элементами скрытого слоя становится почти идентичным правилу обновления для выходного слоя:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

Таким образом, процесс обратного распространения можно кратко описать, как показано ниже.

- Вычислить значения  $\Delta$  для выходных элементов с использованием наблюдаемой ошибки.
- Начиная с выходного слоя, повторять следующие действия для каждого слоя в сети до тех пор, пока не будет достигнут самый первый скрытый слой:
  - распространять значения  $\Delta$  обратно к предыдущему слою;

- обновлять веса между этими двумя слоями.

Подробный алгоритм приведен в листинге 20.2.

#### Листинг 20.2. Алгоритм обратного распространения для обучения в многослойных сетях

---

```

function Back-Prop-Learning(examples, network) returns нейронная сеть
    inputs: examples, множество примеров, для каждого из которых заданы
        входной вектор x и выходной вектор y
        network, многослойная сеть с L слоями, весами  $W_{j,i}$  и
        функцией активации  $g$ 

    repeat
        for each e in examples do
            for each узел j in выходной слой do  $a_j \leftarrow x_j[e]$ 
            for  $\ell = 2$  to L do
                 $in_i \leftarrow \sum_j W_{j,i} a_j$ 
                 $a_i \leftarrow g(in_i)$ 
                for each узел i in выходной слой do
                     $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$ 
                for  $\ell = L-1$  to 1 do
                    for each узел j in слой  $\ell$  do
                         $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$ 
                    for each узел i in слой  $\ell+1$  do
                         $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$ 
            until не достигается некоторый критерий останова
    return Neural-Net-Hypothesis(network)

```

---

Теперь исходя из основных принципов выведем уравнения обратного распространения к радости читателей, интересующихся математикой. Квадратичная ошибка на одном примере определяется следующим образом:

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

где сумма вычисляется по всем узлам выходного слоя. Чтобы определить градиент по отношению к конкретному весу  $W_{j,i}$  в выходном слое, достаточно развернуть только выражение для активации  $a_i$ , поскольку все другие термы в этой операции суммирования не зависят от  $W_{j,i}$ :

$$\begin{aligned}
\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\
&= -(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} \left( \sum_j W_{j,i} a_j \right) \\
&= -(y_i - a_i) g'(in_i) a_j = -a_j \Delta_i
\end{aligned}$$

где  $\Delta_i$  определено, как указано выше. Чтобы получить градиент по отношению к весам  $W_{k,j}$  связей, соединяющих входной слой со скрытым слоем, необходимо по-прежнему вычислять всю сумму по  $i$ , поскольку изменение в значениях  $W_{k,j}$  могут

повлиять на каждое выходное значение  $a_i$ . При этом придется также развертывать выражение для активаций  $a_j$ . Ниже показан ход вывода формулы во всех подробностях, поскольку любопытно понаблюдать за тем, как оператор производной распространяется в обратном направлении через сеть:

$$\begin{aligned}
 \frac{\partial E}{\partial w_{k,j}} &= - \sum_i (y_i - a_i) \frac{\partial a_i}{\partial w_{k,j}} = - \sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial w_{k,j}} \\
 &= - \sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial w_{k,j}} = - \sum_i \Delta_i \frac{\partial}{\partial w_{k,j}} \left( \sum_j w_{j,i} a_j \right) \\
 &= - \sum_i \Delta_i w_{j,i} \frac{\partial a_j}{\partial w_{k,j}} = - \sum_i \Delta_i w_{j,i} \frac{\partial g(in_j)}{\partial w_{k,j}} \\
 &= - \sum_i \Delta_i w_{j,i} g'(in_j) \frac{\partial in_j}{\partial w_{k,j}} \\
 &= - \sum_i \Delta_i w_{j,i} g'(in_j) \frac{\partial}{\partial w_{k,j}} \left( \sum_k w_{k,j} a_k \right) \\
 &= - \sum_i \Delta_i w_{j,i} g'(in_j) a_k = -a_k \Delta_j
 \end{aligned}$$

где  $\Delta_j$  определено, как показано выше. Таким образом, получено то же правило обновления, которое было сформулировано выше на основании интуитивных соображений. Очевидно также, что этот процесс может быть продолжен применительно к сетям, имеющим больше одного скрытого слоя, и это соображение является обоснованием для общего алгоритма, приведенного в листинге 20.2.

Внимательно ознакомившись со всеми этими математическими выкладками (или пропустив их), рассмотрим, какую производительность показывает сеть с одним скрытым слоем при решении задачи с рестораном. На рис. 20.24 показаны две кривые. Первая из них представляет собой **кривую обучения**, которая характеризует изменение среднеквадратичной ошибки на заданном обучающем множестве из 100 примеров для задачи с рестораном в процессе обновления весов. Эта кривая демонстрирует, что параметры сети действительно сходятся, позволяя достичь идеального согласования с обучающими данными. Вторая кривая представляет собой стандартную кривую обучения для данных о ресторане. Нейронная сеть позволяет достичь в процессе обучения хороших результатов, хотя и не совсем с такой скоростью, как при обучении дерева решений; по-видимому, в этом нет ничего удивительного, хотя бы потому, что сами эти данные были сформированы с помощью простого дерева решений.

Безусловно, нейронные сети способны решать гораздо более сложные задачи обучения, хотя и следует отметить, что для создания подходящей сетевой структуры и достижения сходимости к величине, достаточно близкой к глобальному оптимуму в пространстве весов, необходимо приложить определенные усилия. Количество приложений нейронных сетей, описанных в опубликованной литературе, исчисляется буквально десятками тысяч. В разделе 20.7 одно из таких приложений рассматривается более подробно.

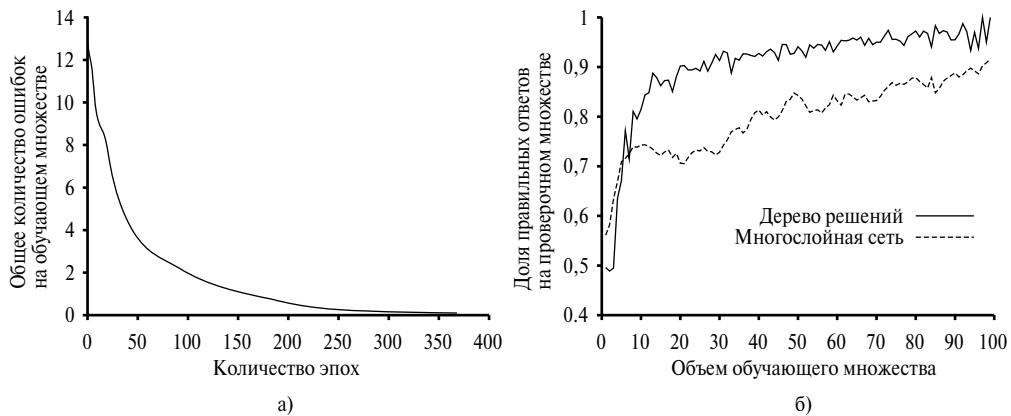


Рис. 20.24. Результаты анализа производительности алгоритма обучения: кривая обучения, показывающая постепенное уменьшение ошибки по мере того, как в течение нескольких эпох происходит модификация весов для заданного множества примеров в задаче с рестораном (а); сопоставление кривых обучения, которое показывает, что обучение дерева решений обеспечивает немногим лучшие результаты, чем обратное распространение в многослойной сети (б)

### Определение в процессе обучения структур нейронных сетей

До сих пор в данной главе рассматривалась проблема определения в процессе обучения весов связей при наличии заданной структуры сети; но, как и в случае с байесовскими сетями, необходимо также знать, как найти наилучшую структуру сети. Если будет выбрана сеть, которая слишком велика, она будет способна запомнить все примеры благодаря формированию большой поисковой таблицы, но не обязательно обеспечит качественное обобщение применительно к входным данным, с которыми еще не сталкивалась<sup>13</sup>. Иными словами, как и все статистические модели, нейронные сети подвержены такому недостатку, как **чрезмерно тщательная подгонка**, когда количество параметров в модели слишком велико. В этом можно убедиться, рассматривая рис. 18.1, где показано, что модели с большим количеством параметров на рис. 18.1, б, в хорошо согласуются со всеми данными, но не обеспечивают такого же хорошего обобщения, как модели с малым количеством параметров, показанные на рис. 18.1, а и г.

Если мы остановимся исключительно на полносвязных сетях, то единственны изменения, которые могут быть внесены в структуру сети, относятся только к количеству скрытых слоев и их размерам. Обычно применяется подход, который предусматривает опробование нескольких вариантов и сохранение наилучшего. Если же решено придерживаться требования по предотвращению **компрометации** проверочного набора, то необходимо использовать методы **перекрестной проверки**, описанные в главе 18. Это означает, что выбор останавливается на такой архитектуре сети, ко-

<sup>13</sup> Было замечено, что очень большие сети все же позволяют обеспечивать хорошее обобщение, при условии, что веса остаются небольшими. Это ограничение равносильно тому, что значения активаций остаются в линейной области сигмоидальной функции  $g(x)$ , где переменная  $x$  близка к нулю. Это, в свою очередь, означает, что сеть действует по аналогии с линейной функцией (упр. 20.17) с гораздо меньшим количеством параметров.

торая обеспечивает наивысшую точность прогнозирования при проверке с помощью испытательных наборов.

Если же решено прибегнуть к использованию сетей, которые не являются полно связными, то необходимо найти какой-то эффективный метод поиска в очень большом пространстве возможных топологий связей. Алгоритм **оптимального повреждения мозга** (optimal brain damage) начинает свою работу с полно связной сети и удаляет из нее связи. После обучения сети в первый раз с помощью подхода на основе теории информации определяется оптимальный выбор связей, которые могут быть удалены. После этого осуществляется повторное обучение сети, и если ее производительность не уменьшается, то этот процесс повторяется. Кроме удаления связей, возможно также удаление элементов, которые не вносят большого вклада в результат.

Было также предложено несколько алгоритмов для формирования большой сети из малой. Один из таких алгоритмов, называемый алгоритмом **заполнения мозаики** (tiling), напоминает алгоритм обучения списков решений. Его идея состоит в том, что процесс начинается с одного элемента, который настраивается наилучшим образом для выработки правильных выходных данных для максимально возможного количества обучающих примеров. Для обработки примеров, которые не могли быть правильно обработаны первым элементом, вводятся дополнительные элементы. Этот алгоритм предусматривает введение только такого количества элементов, которое требуется для охвата всех примеров.

## 20.6. ЯДЕРНЫЕ МАШИНЫ

Приведенное выше описание нейронных сетей не дает ответа на одну дилемму. Однослойные сети позволяют использовать простой и эффективный алгоритм обучения, но обладают лишь очень ограниченной выразительной мощью, поскольку способны определять в процессе обучения только линейные границы между решениями в пространстве входов. Многослойные сети, с другой стороны, являются гораздо более выразительными (они способны представлять нелинейные функции общего вида), но задача их обучения становится очень сложной из-за большого количества локальных минимумов и высокой размерности пространства весов. В этом разделе рассматривается относительно новое семейство методов обучения, основанных на использовании **машин поддерживающих векторов** (Support Vector Machine — SVM), или, в более общем смысле, **ядерных машин** (kernel machine). Ядерные машины позволяют в определенной степени воспользоваться наилучшими свойствами и однослойных, и многослойных сетей. Это означает, что в методах, основанных на их использовании, предусмотрен эффективный алгоритм обучения, а сами они позволяют представить сложные, нелинейные функции.

Полное описание ядерных машин выходит за рамки данной книги, но мы можем проиллюстрировать их основную идею на примере. На рис. 20.25, *a* показано двухмерное пространство входов, определяемое атрибутами  $\mathbf{x} = (x_1, x_2)$ , в котором положительные примеры ( $y=+1$ ) находятся внутри круга, а отрицательные примеры ( $y=-1$ ) — вне его. Очевидно, что для данной задачи не существует линейного разделителя. А теперь предположим, что входные данные выражены иначе, с помощью каких-то вычислимых характеристик, т.е. что каждый вектор входных данных  $\mathbf{x}$  отображен на новый вектор

тор значений характеристик,  $F(\mathbf{x})$ . В частности, предположим, что используются следующие три характеристики:

$$\begin{aligned}f_1 &= x_1^2 \\f_2 &= x_2^2 \\f_3 &= \sqrt{2}x_1x_2\end{aligned}\tag{20.16}$$

Вскоре будет показано, как получены эти выражения, а пока просто рассмотрим, что происходит. На рис. 20.25, *a* показаны данные в этом новом, трехмерном пространстве, определенном тремя характеристиками; очевидно, что данные в этом пространстве являются линейно разделимыми! Такой подход действительно является достаточно общим: если данные отображаются на пространство с достаточно большим количеством размерностей, то они всегда могут быть преобразованы в линейно разделимую форму. В данном случае использовались только три размерности<sup>14</sup>, но если бы количество точек данных было равно  $N$ , то, за исключением частных случаев, они всегда являются разделимыми в пространстве с  $N-1$  размерностями или больше (упр. 20.21).

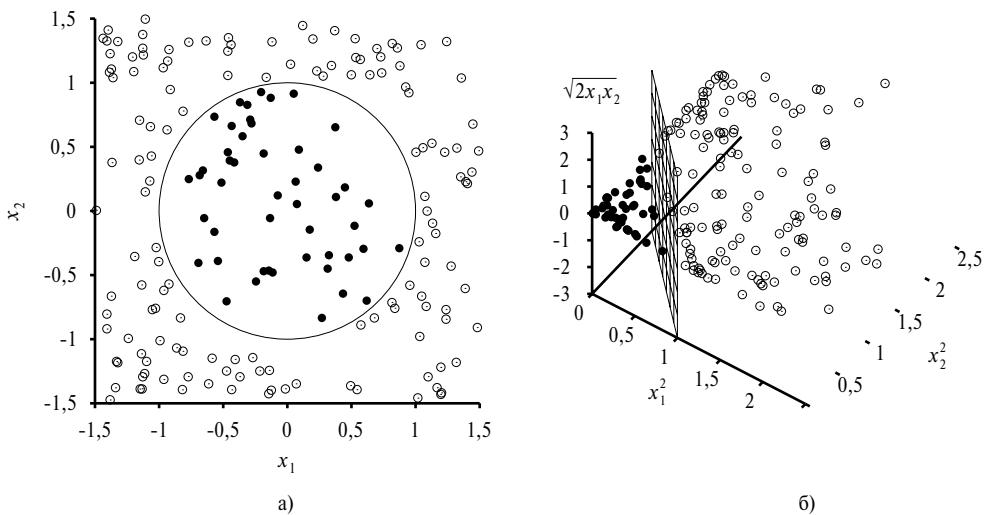


Рис. 20.25. Пример применения преобразования: двухмерная задача обучения с положительными примерами, показанными в виде черных кружков, и отрицательными примерами, обозначенными белыми кружками. Показана также истинная граница решений,  $x_1^2 + x_2^2 \leq 1$  (*a*); те же данные после отображения на трехмерное пространство входов  $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$ . Круглая граница решений, показанная на рис. 20.25, *a*, в трехмерном пространстве становится линейной границей решения (*b*)

Итак, можно ли считать, что на этом проблема исчерпана? Достаточно ли просто подготовить целый ряд расчетных характеристик, а затем найти линейный разделитель в соответствующем многомерном пространстве? К сожалению, все не так просто. Напомним, что линейный разделитель в пространстве с  $d$  размерностями определяется уравнением с  $d$  параметрами, поэтому возникает серьезная опасность

<sup>14</sup> Читатель мог заметить, что достаточно было использовать только характеристики  $f_1$  и  $f_2$ , но трехмерное отображение позволяет лучше проиллюстрировать эту идею.

чрезмерно тщательной подгонки данных, если  $d \approx N$ , т.е. приблизительно равно количеству точек данных. (Такая ситуация аналогична чрезмерно тщательной подгонке к данным с помощью полинома высокой степени, о чем шла речь в главе 18.) По этой причине ядерные машины обычно находят оптимальный линейный разделитель, т.е. такой разделитель, который имеет наибольший **край** между ним и положительными примерами, с одной стороны, и отрицательными примерами, с другой (рис. 20.26). Можно показать, используя аргументы, основанные на теории вычислительного обучения (см. раздел 18.5), что такой разделитель обладает желаемыми свойствами с точки зрения возможности надежного обобщения новых примеров.

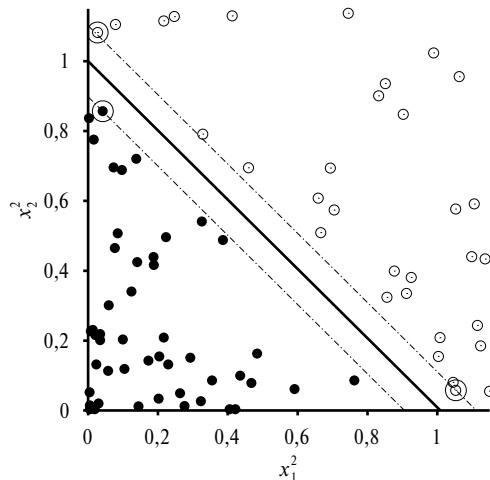


Рис. 20.26. Замыкание оптимального разделителя, показанного на рис. 20.25, б, спроектированное на первые две размерности. Разделитель показан в виде жирной линии, а ближайшие точки (поддерживающие векторы) обозначены кружками. Край представляет собой разделение между положительными и отрицательными примерами

Но как найти такой разделитель? Оказалось, что эта задача представляет собой задачу оптимизации из области **квадратичного программирования**. Предположим, что имеются примеры  $\mathbf{x}_i$  с классификациями  $y_i = \pm 1$  и необходимо найти оптимальный разделитель в пространстве входов; в таком случае задача квадратичного программирования сводится к поиску значений параметров  $\alpha_i$ , которые максимизируют следующее выражение с учетом ограничений  $\alpha_i \geq 0$  и  $\sum_i \alpha_i y_i = 0$ :

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (20.17)$$

Хотя для понимания излагаемого материала не обязательно знакомиться с тем, как было выведено данное выражение, следует отметить, что оно имеет два важных свойства. Во-первых, это выражение имеет единственный глобальный максимум, который может быть найден с помощью эффективных методов. Во-вторых, **данные входят в это выражение только в форме точечных произведений пар точек**. Утверждение о существовании

такого второго свойства является также справедливым применительно к уравнению для самого разделителя; как только будут вычислены оптимальные значения  $\alpha_i$ , появляется возможность вычислить следующее выражение:

$$h(\mathbf{x}) = \text{sign}\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i)\right) \quad (20.18)$$

Последнее важное свойство оптимального разделителя, определяемого этим уравнением, заключается в том, что веса  $\alpha_i$ , связанные с каждой точкой данных, являются нулевыми, кроме тех точек, которые являются ближайшими к разделителю; эти точки называются **поддерживающими векторами** (они получили такое название потому, что на них как бы “опирается” разделяющая гиперплоскость). Поскольку количество поддерживающих векторов обычно намного меньше, чем количество точек данных, результирующее количество параметров, определяющих оптимальный разделитель, так же намного меньше, чем  $N$ .

Итак, обычно не стоит рассчитывать на то, что удастся найти линейный разделитель в пространстве входов  $\mathbf{x}$ , но легко показать, что можно найти линейные разделители в многомерном пространстве характеристик  $F(\mathbf{x})$ , просто заменив точечное произведение  $\mathbf{x}_i \cdot \mathbf{x}_j$  в уравнении 20.17 произведением  $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$ . В этой операции, отдельно взятой, нет ничего необычного (поскольку требуемый эффект может быть достигнут путем замены  $\mathbf{x}$  на  $F(\mathbf{x})$  в любом алгоритме обучения), но точечное произведение обладает некоторыми особыми свойствами. Как оказалось, значение  $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$  часто можно вычислить без предварительного вычисления характеристики  $F$  для каждой точки. В рассматриваемом трехмерном пространстве, определяемом уравнением 20.16, с помощью несложных алгебраических преобразований можно показать, что справедливо следующее соотношение:

$$F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

Выражение  $(\mathbf{x}_i \cdot \mathbf{x}_j)^2$  называется **ядерной функцией** и обычно записывается как  $K(\mathbf{x}_i, \mathbf{x}_j)$ . В контексте проблематики ядерных машин под этим подразумевается любая функция, которая может быть применена к парам выходных данных для вычисления точечных произведений в некотором соответствующем пространстве характеристик. Таким образом, утверждение, приведенное в начале данного абзаца, можно сформулировать следующим образом: линейные разделители в многомерном пространстве характеристик  $F(\mathbf{x})$  можно найти, заменив выражение  $\mathbf{x}_i \cdot \mathbf{x}_j$  в уравнении 20.17 ядерной функцией  $K(\mathbf{x}_i, \mathbf{x}_j)$ . Таким образом, может быть организовано обучение в многомерном пространстве, но для этого придется вычислять только значения ядерных функций, а не значения полного списка характеристик для каждой точки данных.

Следующий этап, который теперь должен стать полностью очевидным, состоит в демонстрации того, что в ядерной функции  $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$  нет ничего особенного, просто она соответствует конкретному пространству характеристик с большим количеством размерностей, а другие ядерные функции соответствуют другим пространствам характеристик. Один из знаменитых результатов в математике, **теорема Мерсера**

[1035], гласит, что любая “приемлемая” ядерная функция<sup>15</sup> соответствует некоторому пространству характеристик. Эти пространства характеристик могут оказаться очень большими даже применительно к таким ядерным функциям, которые выглядят совершенно “невинно”. Например, ~~полиномиальная ядерная функция~~,  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$ , соответствует пространству характеристик, количество размерностей которого определяется экспоненциальной зависимостью от  $d$ . Поэтому использование ядерных функций, подобных приведенной в уравнении 20.17, ~~обеспечивает эффективный поиск оптимальных линейных разделителей в пространствах характеристик с миллиардами (а в некоторых случаях с бесконечным количеством) размерностей~~. Полученные в результате линейные разделители после их обратного отображения на первоначальное пространство входов могут соответствовать произвольно сложным, нелинейным границам между положительными и отрицательными примерами.

Как было упомянуто в предыдущем разделе, ядерные машины превосходят все другие способы распознавания рукописных цифр; кроме того, они быстро находят применение и в других приложениях, особенно в тех, которые отличаются большим количеством входных характеристик. В составе этого процесса было разработано много новых ядерных функций, позволяющих работать со строками, деревьями и другими нечисловыми типами данных. Было также отмечено, что метод ядерных функций может применяться не только в алгоритмах обучения, которые находят оптимальные линейные разделители, но и в любых других алгоритмах, которые можно переформулировать применительно к использованию только точечных произведений пар точек данных, как в уравнениях 20.17 и 20.18. После выполнения такого преобразования точечное произведение заменяется ядерной функцией, что приводит к получению версии того же алгоритма, ~~преобразованной в ядерную форму~~. Кроме всего прочего, такое преобразование можно легко применить для обучения с к ближайшими соседними точками и для обучения персептрона.

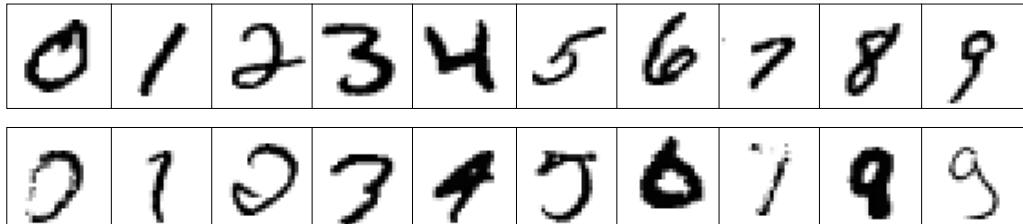
## 20.7. ПРАКТИЧЕСКИЙ ПРИМЕР: РАСПОЗНАВАНИЕ РУКОПИСНЫХ ЦИФР

Задача распознавания рукописных цифр является важной во многих приложениях, включая автоматизированную сортировку почты по почтовому коду, автоматизированное чтение чеков и налоговых деклараций, а также ввод данных для портативных компьютеров. В этой области достигнут быстрый прогресс отчасти благодаря применению лучших алгоритмов обучения и отчасти благодаря наличию лучших обучающих наборов. В Национальном институте стандартов и технологий (National Institute of Standards and Technology — NIST) США был создан архив, представляющий собой базу данных из 60 000 рукописных цифр с расшифровками (с так называемой разметкой), каждая из которых задана в виде изображения, состоящего из  $20 \times 20 = 400$  пикселов с восьмибитовой кодировкой уровня серого. Распознавание

---

<sup>15</sup> Здесь под понятием “приемлемой” ядерной функции подразумевается, что матрица  $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$  является положительной и определенной; см. приложение А.

цифр из этой базы данных стало одной из стандартных эталонных задач для сравнения новых алгоритмов обучения. Некоторые примеры цифр показаны на рис. 20.27.



*Рис. 20.27. Примеры из базы данных рукописных цифр института NIST. Верхний ряд: примеры цифр 0–9, которые несложно распознать; нижний ряд: более трудные примеры тех же цифр*

Для решения этой задачи было опробовано много разных подходов к обучению. Одним из первых и, по-видимому, самых простых стал классификатор по трем ближайшим соседним точкам, преимуществом которого является также то, что он не требует затрат времени на обучение. Но поскольку он представляет собой алгоритм, основанный на использовании памяти, то должен хранить все 60 000 изображений, поэтому показывает низкую производительность на этапе прогона. При использовании такого классификатора была достигнута частота ошибок при распознавании проверочного набора, равная 2,4%.

Кроме того, для решения этой задачи была разработана нейронная сеть с одним скрытым слоем, состоящая из 400 входных элементов (по одному на каждый пиксель) и 10 выходных элементов (по одному на каждый класс). С использованием перекрестной проверки было обнаружено, что наилучшую производительность обеспечивают примерно 300 скрытых элементов. Поскольку была предусмотрена полная связность между слоями, общее количество весов составляло 123300. Эта сеть позволила достичь уровня частоты ошибок 1,6%.

Чтобы можно было воспользоваться структурой самой задачи (тем, что вход состоит из пикселов в двухмерном массиве и что небольшие изменения в положении или наклоне изображения не имеют значения), был разработан ряд специализированных нейронных сетей, называемых LeNet. Каждая сеть имела входной слой из  $32 \times 32$  элементов, по которым  $20 \times 20$  пикселов отцентровывались таким образом, чтобы каждый входной элемент был представлен с помощью локальной окрестности пикселов. За ними следовали три слоя скрытых элементов. Каждый слой состоял из нескольких гиперплоскостей из  $p \times p$  массивов, где значение  $p$  было меньше, чем в предыдущем слое, с тем чтобы сеть сокращала объем выборки входных данных, и где веса каждого элемента в гиперплоскости ограничивались идентичными весами, с тем чтобы эта гиперплоскость действовала как детектор какой-то характеристики: оно позволяла выделять некоторую характеристику, такую как длинная вертикальная линия или короткая полукруглая дуга. Выходной слой включал 10 элементов. Было опробовано много версий такой архитектуры; в одной из наиболее успешных версий применялись скрытые слои, состоящие соответственно из 768, 192 и 30 элементов. Обучающий набор был дополнен путем применения аффинных преобразований к фактическим входным данным — сдвиг, поворот на небольшой угол и масштабирование изображений (безусловно, эти преобразования должны были

быть небольшими, поскольку иначе цифра 6 могла бы превратиться в 9!). Наилучшим значением частоты ошибок, достигнутым с помощью сетей LeNet, было 0,9%.

В усиленной нейронной сети комбинировались три копии архитектуры LeNet, притом что вторая обучалась на смеси образцов, которые первая распознавала с ошибками 50%, а третья обучалась на образцах, для которых не было достигнуто согласование с помощью первых двух копий. Во время проверки проводилось голосование между тремя сетями с назначенными с помощью их весами для каждой из десяти цифр, а полученные оценки складывались для определения победителя. Частота ошибок при обработке проверочного набора составляла 0,7%.

**МашинаНаименование поддерживаемых векторов** (см. раздел 20.6) с 25000 поддерживающих векторов достигла частоты ошибок 1,1%. Это — замечательное достижение, поскольку метод SVM, как и простой подход с использованием ближайших соседних точек, почти не потребовал размышлений или неоднократных экспериментов со стороны разработчика, но вместе с тем позволил сразу же приблизиться к производительности сетей LeNet, на создание которых ушли годы интенсивных разработок. Разумеется, в машине поддерживающих векторов не используются данные о структуре задачи, поэтому она действовала бы с таким же успехом, если бы те же пиксели были представлены после применения к ним какой-то перестановки.

❖ **Виртуальная машина поддерживающих векторов** начинает работу с обычной машины SVM, а затем совершенствует ее с помощью метода, позволяющего воспользоваться данными о структуре задачи. В этом подходе не разрешается использовать произведения всех пар пикселов — вместо этого в основном применяются ядерные функции, сформированные с помощью пар ближайших пикселов. В нем также была предусмотрена возможность дополнять обучающий набор преобразованными вариантами примеров, как и в проекте LeNet. Виртуальная машина SVM достигла наилучшего показателя частоты ошибок, зарегистрированного до настоящего времени, который равен 0,56%.

**Согласование с формой** — это метод из области машинного зрения, который используется для выравнивания соответствующих частей двух различных изображений объектов (см. главу 24). Идея этого метода состоит в том, что выбирается множество точек каждого из двух изображений, а затем для каждой точки из первого изображения с помощью вычислений определяется, какая точка соответствует ей во втором изображении. После этого на основании полученных данных о выравнивании вычисляется преобразование между изображениями, которое позволяет определить значение критерия расстояния между этими изображениями. Такой критерий расстояния является более обоснованным по сравнению с простым подсчетом количества различающихся пикселов, и, как оказалось, очень высокую производительность показывает алгоритм с тремя ближайшими соседними точками, в котором используется этот критерий расстояния. После обучения только на 20000 из 60000 цифр и с использованием 100 выборочных точек в расчете на каждое изображение, выделенных с помощью детектора края Кэнни, классификатор с согласованием формы достиг частоты ошибок при обработке проверочного набора, равной 0,63%.

По некоторым оценкам, люди допускают ошибки при решении задачи распознавания рукописных цифр с частотой примерно 0,2%. Но этим данным не следует полностью доверять, поскольку отнюдь не проводилась такая исчерпывающая проверка способностей людей, как самих алгоритмов машинного обучения. На анало-

гичном наборе данных, состоящем из цифр, полученных из почтовой службы США, частота ошибок, допущенных людьми, составляла примерно 2,5%.

Цифры, приведенные ниже, представляют собой итоговые данные по частоте ошибок, производительности на этапе прогона, потребностям в памяти и продолжительности времени обучения для семи описанных здесь алгоритмов. К этим данным добавлен еще один критерий — процентное количество цифр, которые должны быть отброшены, чтобы можно было достичь частоты ошибок 0,5%. Например, если при использовании алгоритма SVM разрешено отбрасывать 1,8% входных данных (т.е. передавать их кому-то другому, чтобы он сделал окончательное заключение), то частота ошибок на оставшихся 98,2% входных данных сокращается с 1,1% до 0,5%.

Итоговые данные по частоте ошибок и некоторых других характеристиках семи методов, описанных в этом разделе, приведены в табл. 20.2.

**Таблица 20.2. Итоговые данные о семи методах распознавания рукописных цифр**

	С тремя ближайшими соединими точками	С 300 скрытыми элементами	LeNet LeNet	Усиленная LeNet	SVM	Виртуальная SVM	Согласование формы
Частота ошибок (проценты)	2,4	1,6	0,9	0,7	1,1	0,56	0,63
Время выполнения (мс/цифра)	1000	10	30	50	2000	200	
Потребность в памяти (Мбайт)	12	0,49	0,012	0,21	11		
Время обучение (суток)	0	7	14	30	10		
Процент отвергнутых примеров, позволяющий достичь ошибки 0,5%	8,1	3,2	1,8	0,5	1,8		

## 20.8. РЕЗЮМЕ

Разработан широкий перечень статистических методов обучения, начиная от простого вычисления средних и заканчивая построением сложных моделей, таких как байесовские и нейронные сети. Эти методы находят широкое применение в компьютерной науке, техническом проектировании, нейробиологии, психологии и физике. В данной главе представлены некоторые основные идеи из этой области и приведена часть математических выкладок. Основные темы, рассматриваемые в этой главе, перечислены ниже.

- В методах **байесовского обучения** задача обучения формулируется как один из видов вероятностного вывода, в котором наблюдения используются для обновления распределений априорных вероятностей по гипотезам. Такой подход представляет собой хороший способ реализации принципа бритвы Оккама, но быстро становится трудно осуществимым при возрастании сложности пространства гипотез.

- В обучении на основе **максимальной апостериорной** вероятности (Maximum A Posteriori — МАР) выбирается единственная гипотеза, наиболее вероятная согласно имеющимся данным. При этом все еще используется распределение априорных вероятностей гипотезы и сам этот метод часто является более легко осуществимым, чем полное байесовское обучение.
- В обучении с учетом **максимального правдоподобия** выбирается гипотеза, которая максимизирует правдоподобие данных; этот метод эквивалентен методу обучения МАР с равномерным распределением априорных вероятностей. В простейших случаях, таких как линейная регрессия и полностью наблюдаемые байесовские сети, решения с учетом максимального правдоподобия можно легко найти в замкнутой форме. Особенно эффективным методом, который хорошо масштабируется, является **наивное байесовское** обучение.
- Если некоторые переменные скрыты, то решения с локальным максимальным правдоподобием можно найти с использованием алгоритма ЕМ. В число приложений соответствующего метода входит кластеризация с помощью смешанных гауссовых распределений, обучение байесовских сетей и скрытых марковских моделей.
- Определение в процессе обучения структур байесовских сетей представляет собой пример метода **выбора модели**. В этом методе обычно предусматривается дискретный поиск в пространстве структур. При этом необходимо каким-то образом обеспечить поиск компромисса между сложностью модели и степенью ее соответствия данным.
- В **моделях на основе экземпляра** распределение представлено с использованием коллекции обучающих экземпляров. Таким образом, количество параметров растет с увеличением размеров обучающего множества. В методах **ближайшей соседней точки** осуществляется поиск экземпляров, ближайших к рассматриваемой точке, а в **ядерных** методах формируется комбинация всех экземпляров, взвешенная по расстоянию.
- **Нейронные сети** позволяют определять в процессе обучения сложные нелинейные функции со многими параметрами. Изучение этих параметров может осуществляться с применением зашумленных данных, а сами нейронные сети используются в тысячах приложений.
- **Перцептрон** — это нейронная сеть с прямым распространением без скрытых элементов, которая способна представить только **линейно разделимые** функции. Если данные являются линейно разделимыми, то для обеспечения точного согласования с данными может использоваться простое правило обновления весов.
- Такие нейронные сети, как **многослойные сети с прямым распространением**, способны представлять любые функции при наличии достаточного количества элементов. В алгоритме **обратного распространения** реализуется метод градиентного спуска в пространстве параметров для минимизации выходной ошибки.

Статистическое обучение продолжает оставаться очень активной областью исследования. Были достигнуты колоссальные успехи как в теории, так и в практике, и уровень знаний поднялся на такую высоту, что теперь существует возможность оп-

ределить в процессе обучения параметры почти любой модели, для которой осуществим точный или приближенный вероятностный вывод.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

---

В ранние годы развития искусственного интеллекта приложения статистических методов обучения были активной областью исследований [421], но отделились от основного направления искусственного интеллекта после того, как работы в этом направлении сосредоточились на символических методах. Тем не менее исследования статистических методов обучения продолжались в различных формах (часть которых явно относилась к вероятностным, а другая — нет) в таких областях, как **распознавание образов** [394] и **информационный поиск** [1347]. Возрождение всеобщего интереса к этой теме началось вскоре после появления моделей байесовских сетей в конце 1980-х годов; приблизительно в то же время начала формироваться статистическая научная трактовка процесса обучения нейронных сетей. В конце 1990-х годов произошло заметное пробуждение интереса к машинному обучению, статистике и нейронным сетям и значительные усилия были сосредоточены на разработке методов создания больших вероятностных моделей на основе данных.

Наивная байесовская модель представляет собой одну из самых старых и наиболее простых форм байесовской сети, которая была впервые описана в 1950-х годах. О том, каково происхождение этой модели, упоминалось в заметках в конце главы 13; частичное описание этой темы приведено в [402]. Программа на основе усиленной формы наивного байесовского обучения стала победителем первого соревнования по интеллектуальному анализу данных на кубок KDD Cup [435]. В [641] приведено превосходное введение в общую проблему обучения байесовской сети. Определение параметров байесовской сети с помощью распределений априорных вероятностей Дирихле для байесовских сетей рассматривалось в [1450]. Многие из этих идей были реализованы в программном пакете Bugs [555], который представляет собой очень мощное инструментальное средство для формулировки и обучения сложных вероятностных моделей. В первых алгоритмах для определения в процессе обучения структур байесовских сетей использовались проверки условной независимости [1191], [1193]. В [1452] разработан исчерпывающий подход и описан пакет Tetrad для обучения байесовских сетей на основе аналогичных идей. Достигнутые с тех пор усовершенствования алгоритмов стали причиной убедительной победы метода обучения байесовской сети в соревновании по интеллектуальному анализу данных на кубок KDD Cup 2001 года [248]. (На этих соревнованиях рассматривалась конкретная задача из биоинформатики с 139351 характеристикой!) Подход к определению в процессе обучения структуры сети, основанный на учете максимального правдоподобия, был разработан Купером и Херковицем [292] и усовершенствован Хекерманом и др. [642]. В [507] указано на то, какое влияние оказывает способ представления локальных распределений условных вероятностей на структуру, определяемую в процессе обучения.

Общая задача определения в процессе обучения параметров вероятностных моделей со скрытыми переменными и недостающими данными была решена с помощью алгоритма EM, предложенного Демпстером [383], который представляет собой обобщение нескольких существующих методов, включая алгоритм Баума–Уэлша для

обучения скрытых марковских моделей [85]. (Сам Демпстер рассматривал ЕМ скорее как схему, а не как алгоритм, поскольку может потребоваться большой объем математической работы, прежде чем появится возможность применить подход на основе ЕМ к новому семейству распределений.) В настоящее время ЕМ представляет собой один из алгоритмов, наиболее широко используемых в науке, а Маклахлан и Кришнан посвятили этому алгоритму и его свойствам целую книгу [1030]. Конкретная задача определения в процессе обучения параметров моделей на основе смешанных распределений, включая смешанные гауссовые распределения, рассматривается в [1509]. В рамках искусственного интеллекта первой успешной системой, в которой использовался алгоритм ЕМ для моделирования смешанных распределений, была система Autoclass [245], [246]. Система Autoclass применялась для решения многих реальных задач научной классификации, включая открытие новых типов звезд на основе спектральных данных [567] и новых классов белков и инtronов в базах данных последовательностей ДНК/белок [708].

Алгоритм ЕМ для обучения байесовских сетей со скрытыми переменными был разработан Лауритценом [892]. Наряду с этим свою эффективность при обучении байесовских сетей, а также динамических байесовских сетей показали методы на основе градиента [1326], [126]. Структурный алгоритм ЕМ был разработан Фридманом [506]. Способность к определению в процессе обучения структуры байесовских сетей тесно связана с проблемой извлечения причинной информации из данных. Эта проблема сводится к поиску ответа на вопрос о том, существует ли возможность определять в процессе обучения структуру байесовских сетей таким образом, чтобы полученная структура сети демонстрировала реальные причинные связи? В течение многих лет статистики избегали анализа этого вопроса, считая, что данные самих наблюдений (в отличие от данных, выработанных в результате экспериментальных попыток) могут предоставить только информацию о корреляции; в конце концов, любые две переменные, которые кажутся взаимосвязанными, могут в действительности испытывать влияние третьего, неизвестного причинного фактора, а не влиять друг на друга непосредственно. Перл [1192] представил убедительные доводы, опровергающие это мнение, и показал, что фактически возникает много ситуаций, в которых причинно-следственные связи можно подтвердить и выявить с помощью формальных средств  **причинной сети** для выражения причин и результатов вмешательства, а также обычных условных вероятностей.

Истоки моделей с использованием ближайших соседних точек прослеживаются по меньшей мере до работы Фикса и Ходжеса [474] и со времени ее появления такие модели считаются стандартным инструментом в статистике и распознавании образов. В искусственном интеллекте они нашли широкое применение под влиянием работы Стенфилла и Вальца [1457], которые исследовали методы адаптирования метрики расстояния к данным. Хости и Тибширани [628] разработали способ локализации метрики применительно к каждой точке пространства в зависимости от распределения данных вокруг этой точки. Эффективные схемы индексации для поиска ближайших соседних точек исследовались в сообществе специалистов по алгоритмам (см., например, [715]). Оценки плотности ядра, называемые также оценками плотности **окна Парцена**, были первоначально исследованы Розенблattом [1305] и Парценом [1178]. С тех пор было опубликовано огромное количество научных работ с результатами исследований свойств различных средств оценки. Исчерпывающее введение в эту тему приведено в [393].

Объем литературы по нейронным сетям велик (до настоящего времени опубликовано примерно 100 000 статей), чтобы всю ее можно было подробно рассмотреть в настоящем разделе. В [299], [300] приведен краткий обзор ранней истории этого направления, начиная с работы Мак-Каллока и Питтса [1017]. В сотрудничестве с Мак-Каллоком и Питтсом работал Норберт Винер, основатель кибернетики и теории управления [1589], который оказал значительное влияние на дальнейшую деятельность многих молодых исследователей, включая Марвина Минского. По-видимому, именно Минский был первым, кто разработал действующую нейронную сеть на основе аппаратных средств; это произошло в 1951 году (см. [1055, с. ix–x]). Между тем в Великобритании У. Росс Эшби (также один из основателей кибернетики; см. [42]), Алан Тьюринг, Грей Уолтер и другие основали клуб *Ratio* (клуб Разума) для “тех, кто был носителем идей Винера еще до появления книги Винера”. В книге Эшби *Design for a Brain* [43], [44] выдвинута идея, что интеллект можно создать с использованием **гомеостатических** устройств, реализующих соответствующие циклы обратной связи для достижения стабильного адаптивного поведения. Тьюринг [1519] написал исследовательский отчет с заглавием *Intelligent Machinery*, который начинается со слов “Я предлагаю исследовать вопрос о том, может ли машина проявлять интеллектуальное поведение” и продолжается в виде описания архитектуры рекуррентной нейронной сети, названной Тьюрингом “неорганизованными машинами В-типа”, и подхода к обучению этих машин. К сожалению, этот отчет оставался неопубликованным до 1969 года и его содержание почти полностью игнорировалось до недавнего времени.

Фрэнк Розенблatt [1302] изобрел современный “персептрон” и доказал теорему сходимости персептрана [1303], хотя его работы оставались в тени чисто математических исследований, выполненных вне контекста нейронных сетей [6], [1093]. Кроме того, некоторые ранние работы в области нейронных сетей были посвящены многослойным сетям, включая **персептроны Гамба** [517] и **мадалины** [1586]. В книге *Learning Machines* [1140] рассматриваются многие из этих ранних работ, а также другие интересные темы. В дальнейшем, в этот ранний период исследований персептранов, интерес к этой теме упал под влиянием книги *Perceptrons* [1054], авторы которой посетовали на отсутствие математической строгости в этой области (но сами авторы в последующем заявили, что они в своей книге просто объяснили причины этого падения интереса). В данной книге указано, что однослойные персептроны способны представить только линейно разделимые понятия, и отмечено отсутствие эффективных алгоритмов обучения для многослойных сетей.

Как свидетельство возрождения интереса к коннекционизму могут рассматриваться статьи в сборнике, выпущенном по материалам конференции в Сан-Диего в 1979 году [655]. Большое внимание исследователей привлекла двухтомная антология *PDP* (Parallel Distributed Processing — параллельная распределенная обработка) [1316] и короткая статья в журнале *Nature* [1317]; фактически количество статей по “нейронным сетям” за период между 1980–1984 и 1990–1994 гг. увеличилось в 200 раз. Анализ нейронных сетей с использованием физической теории магнитных спиновых стекол, приведенный в [26], упрочил связи между статистической механикой и теорией нейронных сетей, предоставляя последнему научному направлению не только полезные математические основы, но и научную респектабельность. Метод обратного распространения был изобретен довольно рано [201], но затем был забыт и снова открыт еще несколько раз [1175], [1579].

Машины поддерживающих векторов впервые были созданы в 1990-х годах [296], а теперь являются темой все более возрастающего количества литературных источников, включая такие учебники, как [309]. Было доказано, что эти машины могут стать очень широко применяемым и эффективным средством решения таких задач, как категоризация текста [738], исследования в области биоинформатики [194] и обработка естественного языка, в частности распознавание рукописных цифр [374]. К примерам связанных с ними методов, в которых также используется “фокус с ядерными функциями” для неявного представления экспоненциального пространства характеристик, относится персепtron с голосованием [283].

Тема вероятностной интерпретации нейронных сетей рассматривалась в нескольких источниках, включая [84] и [185]. Роль сигмоидальной функции описана в [745]. Метод байесовского обучения параметрам для нейронных сетей был предложен Маккеем [965], а его дальнейшее исследование проведено Нилом [1118]. Способность нейронных сетей представлять функции была исследована Цыбенко [316], [317], который показал, что двух скрытых слоев достаточно для представления любой функции, а одного скрытого слоя достаточно для представления любой непрерывной функции. Метод “оптимального повреждения мозга”, предназначенный для удаления бесполезных связей, изложен в [903], а в [1409] показано, как удалять неужные элементы. Алгоритм заполнения мозаики, предназначенный для наращивания размеров структур, предложен в [1037]. В [904] приведен обзор целого ряда алгоритмов распознавания рукописных цифр. С тех пор были опубликованы сведения о достигнутых успехах в области уменьшения частоты ошибок в [98] с помощью алгоритма согласования с формой и в [374] — с помощью алгоритма для виртуальных поддерживающих векторов.

Проблемы сложности обучения нейронных сетей рассматривались исследователями, занимающимися теорией вычислительного обучения. Первые вычислительные результаты были получены Джаддом [753], который показал, что общая задача поиска множества весов, совместимых с множеством примеров, является NP-полной, даже при очень ограничительных предположениях. Некоторые из первых результатов, касающихся выборочной сложности, были получены Баумом и Хаусслером [82], которые показали, что количество примеров, требуемых для эффективного обучения, растет примерно пропорционально  $\log W$ , где  $W$  — количество весов<sup>16</sup>. С тех пор была разработана гораздо более совершенная теория [34], в том числе получен важный результат, показывающий, что репрезентативная способность сети зависит не только от количества весов, но и от их величины.

Наиболее широко применяемой разновидностью нейронных сетей из тех, которые не рассматривались в данной книге, является сеть  с **радиальной базисной функцией**, или сокращенно RBF (Radial Basis Function). В радиальной базисной функции объединяется взвешенная коллекция ядерных функций (разумеется, обычно гауссовых распределений) для осуществления функциональной аппроксимации. Обучение сетей RBF может проводиться в два этапа: вначале с помощью подхода на основе неконтролируемой кластеризации происходит определение в процессе обучения параметров гауссовых распределений (математических ожиданий

<sup>16</sup> Этот результат примерно соответствует “правилу дяди Берни”. Это правило получило название в честь Берни Видроу, который рекомендовал использовать примерно в десять раз больше примеров по сравнению с весами.

и дисперсий), как описано в разделе 20.3. На втором этапе определяются относительные веса гауссовых распределений. Они составляют систему линейных уравнений, которые, как известно, можно решить непосредственно. Поэтому два этапа обучения RBF предоставляют важное преимущество: первый этап является неконтролируемым, и поэтому для него не требуются размеченные обучающие данные, а второй этап, хотя и контролируемый, характеризуется высокой эффективностью. Подробные сведения приведены в [133].

В этой главе упоминались, но подробно не рассматривались **рекуррентные сети**. По-видимому, наиболее изученным классом рекуррентных сетей являются **сети Хопфилда** [674]. В них используются двунаправленные связи с симметричными весами (т.е. элементы с  $W_{i,j} = W_{j,i}$ ), все элементы являются одновременно входными и выходными, функция активации,  $g$ , представляет собой знаковую функцию, а уровни активации могут принимать только значения  $\pm 1$ . Сеть Хопфилда функционирует как **ассоциативная память**: после обучения сети на множестве примеров новый стимул вызывает установление в сети образа активации, соответствующего тому примеру в обучающем множестве, который наиболее близко напоминает этот новый стимул. Например, если обучающее множество состоит из набора фотографий и новым стимулом является небольшой фрагмент одной из фотографий, то уровни активации сети воспроизводят фотографию, из которой был взят этот фрагмент. Следует отметить, что оригинальные фотографии не хранятся отдельно в сети; каждый вес представляет собой результат частичного кодирования всех фотографий. Одним из наиболее интересных теоретических результатов является то, что сети Хопфилда могут надежно хранить вплоть до  $0,138 N$  обучающих примеров, где  $N$  — количество элементов в сети.

В **машинах Больцмана** [657], [658] также используются симметричные веса, но предусматриваются скрытые элементы. Кроме того, в них применяется стохастическая функция активации, такая что вероятность появления на выходе 1 определяется некоторой функцией от общего взвешенного входа. Поэтому машины Больцмана подвержены переходам между состояниями, которые напоминают поиск с эмуляцией отжига (см. главу 4), применительно к конфигурации, которая наилучшим образом аппроксимирует обучающее множество. Как оказалось, машины Больцмана очень тесно связаны с частным случаем байесовских сетей, оценка параметров которых осуществляется с помощью алгоритма стохастического моделирования (см. раздел 14.5).

Первое приложение идей, лежащих в основе ядерных машин, было разработано Айзermanом и др. [11], но полная разработка теории этих машин под названием **машин поддерживающих векторов** была выполнена Владимиром Вапником и его коллегами [156], [1537]. Строгие введение в эту тематику приведены в [309] и [1364]; описание, более удобное для чтения, приведено в статье для журнала *AI Magazine*, написанной Кристианини и Шёлкопфом [308].

В материалах этой главы собраны вместе работы из области статистики, распознавания образов и нейронных сетей, поэтому излагаемые в ней сведения были освещены в литературе много раз многими способами. К хорошим учебникам по байесовской статистике относятся [101], [377] и [534]. В [629] приведено превосходное введение в методы статистического обучения. Классическим учебником по классификации образов в течение многих лет была книга [421], которая недавно была переиздана в обновленном виде [422]. Ведущими учебниками по нейронным сетям яв-

ляются [133] и [1290]. Область вычислительной неврологии рассматривается в [339]. Наиболее важной конференцией по нейронным сетям и относящимся к ним темам является ежегодная конференция *NIPS* (Neural Information Processing Conference), труды которой публикуются в виде серии *Advances in Neural Information Processing Systems*. Статьи по обучению байесовских сетей появляются также в трудах конференций *Uncertainty in AI* и *Machine Learning*, а также нескольких конференций по статистике. К числу журналов, посвященных нейронным сетям, относятся *Neural Computation*, *Neural Networks* и *IEEE Transactions on Neural Networks*.

## УПРАЖНЕНИЯ

- 20.1.** Данные, которые использовались для графика, приведенного на рис. 20.1, можно рассматривать как сформированные с помощью гипотезы  $h_5$ . Для каждой из остальных четырех гипотез сформируйте набор данных с длиной 100 и вычертите соответствующие графики для  $P(h_i | d_1, \dots, d_m)$  и  $P(D_{m+1} = \text{lime} | d_1, \dots, d_m)$ . Прокомментируйте полученные вами результаты.
- 20.2.** Повторите упр. 20.1, но на этот раз нанесите на графики значения  $P(D_{m+1} = \text{lime} | h_{\text{MAP}})$  и  $P(D_{m+1} = \text{lime} | h_{\text{ML}})$ .
- 20.3.** Предположим, что для Анны полезности вишневого и лимонного леденцов равны  $c_A$  и  $\ell_A$ , а для Боба эти полезности равны  $c_B$  и  $\ell_B$  (но после того как Анна развертывает какую-то конфету, Боб эту конфету не покупает). Предполагается, что если Боб любит лимонные леденцы гораздо больше чем Анна, то было бы разумным решением со стороны Анны продать Бобу свой пакет с конфетами после того, как она приобретет достаточную уверенность в наличии в этом пакете большого количества лимонных леденцов. С другой стороны, если Анна в процессе анализа содержимого пакета разворачивает слишком много конфет, стоимость пакета уменьшается, поскольку Боб не платит за развернутые конфеты. Обсудите задачу определения оптимальной точки, в которой Анна должна продавать свой пакет. Определите ожидаемую полезность этой оптимальной процедуры с учетом распределения априорных вероятностей, описанного в разделе 20.1.
- 20.4.** Два статистика попали на прием к врачу, который сообщил им одинаковый прогноз: с вероятностью 40% расстройство их здоровья вызвано смертельным заболеванием *A*, а с вероятностью 60% оно вызвано тяжелым заболеванием *B*. К счастью, есть лекарства и от заболевания *A*, и от заболевания *B*, которые являются недорогими, эффективными на 100% и не вызывающими побочных эффектов. Этим статистикам предоставлена возможность выбрать для себя один из вариантов дальнейших действий — принимать одно из этих лекарств, оба эти лекарства или ни одного из этих лекарств. Как поступит первый статистик, который является убежденным сторонником байесовского подхода? А как поступит второй статистик, который всегда использует гипотезу с максимальным правдоподобием?

Врач провел определенные исследования и обнаружил, что заболевание *B* фактически протекает в двух вариантах (правостороннее заболевание *B* и ле-

востороннее заболевание  $B$ ), которые являются равновероятными и одинаково хорошо излекиваются с помощью лекарства против заболевания  $B$ . Теперь количество гипотез стало равным трем; как поступят эти два статистика?

- 20.5. Объясните, как применить метод усиления, описанный в главе 18, для наивного байесовского обучения. Проверьте производительность результирующего алгоритма на задаче обучения с рестораном.
- 20.6. Рассмотрим  $m$  точек данных  $(x_j, y_j)$ , где координаты  $y_j$  вырабатываются на основании координат  $x_j$  в соответствии с моделью линейного гауссова распределения, приведенной в уравнении 20.5. Найдите значения  $\theta_1$ ,  $\theta_2$  и  $\sigma$ , которые максимизируют условное логарифмическое правдоподобие этих данных.
- 20.7. Рассмотрим модель зашумленного ИЛИ для лихорадки, описанную в разделе 14.3. Объясните, как применить обучение с учетом максимального правдоподобия для согласования параметров такой модели с множеством полных данных. (*Подсказка.* Используйте цепное правило для частичных производных.)
- 20.8. В данном упражнении исследуются свойства бета-распределения, которое определено в уравнении 20.6.
  - a) Выполнив интегрирование по отрезку  $[0, 1]$ , покажите, что константа нормализации для распределения  $\text{beta}[a, b]$  задается выражением  $\alpha = \Gamma(a+b)/\Gamma(a)\Gamma(b)$ , где  $\Gamma(x)$  — **гамма-функция**, определяемая выражением  $\Gamma(x+1) = x\cdot\Gamma(x)$ , а  $\Gamma(1) = 1$  (для целого числа  $x$  выражение  $\Gamma(x+1) = x!$ ).
  - б) Покажите, что математическое ожидание равно  $a/(a+b)$ .
  - в) Найдите моду (моды) — наиболее вероятное значение (значения)  $\theta$ .
  - г) Опишите вид распределения  $\text{beta}[\varepsilon, \varepsilon]$  для очень малого значения  $\varepsilon$ . Что происходит при обновлении такого распределения?
- 20.9. Рассмотрим произвольную байесовскую сеть, полный набор данных для этой сети и правдоподобие этого набора данных согласно этой сети. Дайте простое доказательство того, что правдоподобие данных не может уменьшиться после добавления новой связи к сети и повторного вычисления значений параметров максимального правдоподобия.
- 20.10. Рассмотрим применение алгоритма EM для определения в процессе обучения параметров сети, приведенной на рис. 20.10, *a*, если даны истинные параметры в уравнении 20.7.
  - а) Объясните, почему алгоритм EM не будет действовать, если в модели имеются только два атрибута, а не три.
  - б) Покажите расчеты для первой итерации алгоритма EM, начиная с уравнения 20.8.
  - в) Что происходит, если применение алгоритма начинается с присваивания всем параметрам одинакового значения  $p$ ? (*Подсказка.* Рекомендуется вначале провести эмпирическое исследование этого вопроса и только после этого выводить общий результат.)
  - г) Запишите выражение для логарифмического правдоподобия табличных данных о конфетах, приведенных в табл. 20.1, с учетом параметров, рас-

считайте частичные производные по отношению к каждому параметру и исследуйте характер фиксированной точки, достигнутой при выполнении упр. 20.10, в.

- 20.11.** Постройте вручную нейронную сеть, которая вычисляет функцию XOR от двух входов. Обязательно укажите, какого рода элементы вы используете.
- 20.12.** Сконструируйте машину поддерживающих векторов, которая вычисляет функцию XOR. При этом для входов и выходов удобнее использовать значения 1 и -1 вместо 1 и 0. Поэтому некоторые примеры могут выглядеть как  $([-1, 1], 1)$  или  $([-1, -1], -1)$ . Обычно принято отображать входное значение  $\mathbf{x}$  на пространство, состоящее из пяти размерностей: двух первоначальных размерностей,  $x_1$  и  $x_2$ , и трех комбинаций размерностей,  $x_1^2$ ,  $x_2^2$  и  $x_1x_2$ . Но для этого упражнения мы будем рассматривать только две размерности,  $x_1$  и  $x_1x_2$ . Нарисуйте в этом пространстве четыре входных точки и разделитель с максимальной шириной края. Каковым является этот край? А теперь снова преобразуйте разделительную линию и нарисуйте ее в первоначальном евклидовом пространстве входов.
- 20.13.** Простой персептрон не способен представить функцию XOR (или, вообще говоря, функцию четности от его входов). Опишите, что происходит с весами в четырехходовом персептроне со ступенчатой функцией, начиная со всех весов, установленных равными 0.1, по мере поступления примеров функции четности.
- 20.14.** Напомним, что, как было сказано в главе 18, существует  $2^{2^n}$  различных булевых функций от  $n$  входов. Какая часть функций из этого общего количества может быть представлена с помощью порогового персептрана?
- 20.15.** Рассмотрим приведенное в табл. 20.3 множество примеров, каждый из которых имеет шесть входов и один целевой выход.

Таблица 20.3. Примеры для упр. 20.15

I <sub>1</sub>	1	1	1	1	1	1	1	0	0	0	0	0	0	0
I <sub>2</sub>	0	0	0	1	1	0	0	1	1	0	1	0	1	1
I <sub>3</sub>	1	1	1	0	1	0	0	1	1	0	0	0	1	1
I <sub>4</sub>	0	1	0	0	1	0	0	1	0	1	1	1	0	1
I <sub>5</sub>	0	0	1	1	0	1	1	0	1	1	0	0	1	0
I <sub>6</sub>	0	0	0	1	0	1	0	1	1	0	1	1	1	0
T	1	1	1	1	1	1	0	1	0	0	0	0	0	0

- a) Примените правило обучения персептрана к этим данным и покажите окончательные веса.
- б) Примените правило обучения дерева решений и покажите результирующее дерево решений.
- в) Прокомментируйте полученные вами результаты.

- 20.16.** Начиная с уравнения 20.13, покажите, что  $\partial L / \partial W_j = Err \times a_j$ .

**20.17.** Предположим, что имеется нейронная сеть с линейными функциями активации. Это означает, что выход каждого элемента определяется некоторой константой  $c$ , умноженной на взвешенную сумму его входов.

- a) Предположим, что эта сеть имеет один скрытый слой. Для данного присваивания весам  $\mathbf{W}$  запишите уравнения для значений элементов в выходном слое как функции от  $\mathbf{W}$  и значений элементов входного слоя  $\mathbf{x}$  без какого-либо явного упоминания в этих выходных данных о скрытом слое. Покажите, что существует сеть без скрытых элементов, которая вычисляет ту же функцию.
- б) Повторите вычисления, описанные в упр. 20.17, a, но на этот раз применительно к сети с любым количеством скрытых слоев. Какой можно сделать вывод в отношении линейных функций активации?

**20.18.** Реализуйте какую-то структуру данных для многослойных нейронных сетей с прямым распространением и не забудьте предусмотреть способ представления информации, необходимой как для прямого вычисления, так и для обратного распространения. Используя эту структуру данных, напишите функцию `Neural-Network-Output`, которая принимает на входе определения некоторого примера и сети, после чего вычисляет соответствующие выходные значения.

**20.19.** Предположим, что обучающее множество содержит только единственный пример, повторенный 100 раз. В 80 из 100 случаев единственным выходным значением является 1, а в остальных 20 случаях таковым является 0. Что предсказывает сеть с обратным распространением для данного примера, при условии, что по нему проведено обучение и достигнут глобальный оптимум? (Подсказка. Чтобы найти глобальный оптимум, необходимо дифференцировать функцию ошибки и приравнять полученное выражение к нулю.)

**20.20.** Сеть, приведенная на рис. 20.23, имеет четыре скрытых узла. Это количество узлов было выбрано фактически произвольно. Проведите систематические эксперименты, чтобы измерить кривые обучения для сетей с различным количеством скрытых узлов. Каковым является оптимальное количество? Было бы возможно использовать метод перекрестной проверки, чтобы найти najleшую сеть еще до получения этих данных?

**20.21.** Рассмотрим задачу разделения  $N$  точек данных на положительные и отрицательные примеры с использованием линейного разделителя. Очевидно, что эту задачу всегда можно выполнить для количества точек  $N=2$  на линии с размерностью  $d=1$ , независимо от того, как размечены эти точки или где они находятся (если только эти две точки не находятся в одном и том же месте).

- a) Покажите, что эту задачу всегда можно выполнить для количества точек  $N=3$  на плоскости с размерностью  $d=2$ , если только эти точки не являются коллинеарными.
- б) Покажите, что эту задачу не всегда можно решить для количества точек  $N=4$  на плоскости с размерностью  $d=2$ .

- в) Покажите, что эту задачу не всегда можно решить для количества точек  $N=4$  в пространстве с размерностью  $d=3$ , если только эти точки не являются копланарными.
- г) Покажите, что эту задачу не всегда можно решить для количества точек  $N=5$  в пространстве с размерностью  $d=3$ .
- д) Амбициозный студент решил доказать, что произвольно расположенные  $N$  точек (но не  $N+1$  точка) являются линейно разделимыми в пространстве с размерностью  $N-1$ . Покажите, что из этого доказательства будет следовать, что **VC-размерность** (см. главу 18) линейных полупространств в пространствах с размерностью  $N-1$  равна  $N$ .

# 21 ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

*В этой главе рассматривается вопрос о том, как агент может учиться на своих успехах и неудачах, учитывая полученные вознаграждения и наказания.*

## 21.1. ВВЕДЕНИЕ

В главах 18 и 20 рассматривались методы обучения, позволяющие определять функции и вероятностные модели на основе примеров, а в этой главе будет описано, каким образом агенты могут определить в процессе обучения, что делать, особенно если нет учителя, сообщающего агенту, какое действие следует предпринять в тех или иных обстоятельствах.

Например, как известно, агент может обучиться игре в шахматы с помощью контролируемого обучения, в котором ему предъявляются примеры игровых ситуаций наряду с наилучшими ходами для этих ситуаций. Но если нет дружелюбного учителя, предоставляющего готовые примеры, то что может сделать агент? Опробуя случайно выбранные ходы, агент может в конечном итоге составить прогностическую модель своей среды, т.е. предсказать, как будет выглядеть доска после того, как он сделает данный конкретный ход, и даже как, скорее всего, ответит противник в такой ситуации. Но при этом возникает следующая проблема: *«без какой-либо обратной связи, говорящей о том, какой ход является хорошим и какой плохим, агент не будет иметь оснований для принятия решения о том, какой ход следует сделать»*. Агент должен знать, что его выигрыш — это благоприятный исход, а проигрыш — неблагоприятный. Обратная связь такого рода называется **вознаграждением**, или **подкреплением**. В играх, подобных шахматам, подкреплениедается только в конце игры. В других вариантах среды вознаграждения могут поступать более часто. В настольном теннисе как вознаграждение может рассматриваться каждое выигранное очко, а при обучении новобранцев способам перемещения ползком достижением становится каждое движение вперед. В инфраструктуре для агентов, рассматриваемой в данной главе, вознаграждение считается частью результатов восприятия, но агент должен быть “настроен” на распознавание этой части как вознаграждения, а не про-

сто как еще одного вида сенсорных входных данных. Например, складывается впечатление, что животные настроены на распознавание боли и голода как отрицательных вознаграждений, а удовольствия и приема пищи — как положительных вознаграждений. Проблемы подкрепления тщательно исследовались специалистами в области психологии животных больше 60 лет.

Понятие вознаграждения было впервые представлено в главе 17, где оно использовалось для определения оптимальных стратегий в **марковских процессах принятия решений** (Markov Decision Process — MDP). Оптимальной является такая стратегия, которая максимизирует ожидаемое суммарное вознаграждение. Задача **обучения с подкреплением** состоит в том, чтобы обеспечить использование наблюдаемых вознаграждений для определения в процессе обучения оптимальной (или почти оптимальной) стратегии для данной среды. Но хотя агент, рассматриваемый в главе 17, имел полную модель среды и знал функцию вознаграждения, в данной главе предполагается отсутствие априорных знаний и о том и о другом. Представьте себе, что вы играете в новую игру, правил которой не знаете; примерно через сто ходов ваш противник объявляет: “Вы проиграли”. В этом состоит вся суть обучения с подкреплением.

Во многих сложных проблемных областях обучение с подкреплением является единственным осуществимым способом, с помощью которого можно провести обучение некоторой программы, чтобы она могла действовать с высокой производительностью. Например, в случае ведения игр для человека является очень трудной задачей предоставление точных и согласованных оценок большого количества позиций, что требуется для определения в процессе обучения функций оценки непосредственно из примеров. Вместо этого программе можно сообщать, когда она выиграла или проиграла, а сама программа может использовать такую информацию для определения с помощью обучения такой функции оценки, которая предоставляла бы достаточно точные оценки вероятности выигрыша из любой конкретной позиции. Аналогичным образом, чрезвычайно трудно запрограммировать агента так, чтобы он научился вести вертолет; но, предоставляя соответствующие отрицательные вознаграждения за столкновение, болтанку или отклонение от заданного курса, можно дать агенту возможность научиться летать на вертолете самостоятельно.

Обучение с подкреплением может рассматриваться как задача, охватывающая всю тематику искусственного интеллекта: агента помещают в какую-то среду и обязывают его обучиться успешно действовать в ней. Поэтому, чтобы объем этой главы не вышел за пределы разумного, в ней будут рассматриваться только простые варианты среды и простые проекты агента. По большей части предполагается, что среда является полностью наблюдаемой, поэтому информация о текущем состоянии поступает с результатами каждого восприятия. С другой стороны, считается, что агент не знает, по каким принципам действует среда или какими являются результаты его действий, поэтому допускается наличие вероятностных результатов действий. В этой главе речь пойдет о трех перечисленных ниже проектах агентов, которые были впервые представлены в главе 2.

- **Агент, действующий с учетом полезности**, определяет с помощью обучения функцию полезности состояний и использует ее для выбора действий, которые максимизируют ожидаемую полезность результата.
- Агент, действующий по принципу **Q-обучения**, определяет с помощью обучения функцию **“действие–значение”**, или Q-функцию, получая сведения

об ожидаемой полезности выполнения данного конкретного действия в данном конкретном состоянии.

- **Рефлексный агент** определяет с помощью обучения стратегию, которая непосредственно отображает состояния в действия.

Агент, действующий с учетом полезности, для принятия решений должен также иметь модель среды, поскольку он должен знать, в какие состояния приведут его выполненные им действия. Например, для того чтобы программа игры в нарды могла использовать функцию оценки для нард, она должна иметь информацию о том, каковыми являются допустимые ходы и как они влияют на позицию в игре. Это — единственный способ, позволяющий применить функцию полезности к результатирующим состояниям. Агент, действующий по принципу Q-обучения, с другой стороны, может сравнивать значения, характеризующие доступные ему варианты действий, без необходимости знать их результаты, поэтому ему не требуется модель среды. Тем не менее агенты, действующие по принципу Q-обучения, не могут прогнозировать будущую ситуацию, поскольку не имеют информации о том, к чему приведут их действия; это может серьезно ограничить способность таких агентов к обучению, как будет описано ниже.

Изложение материала этой главы начинается в разделе 21.2 с описания **пассивного обучения**, в котором стратегия агента остается неизменной, а задача состоит в том, чтобы определить с помощью обучения полезности состояний (или пар “состояние–действие”); для этого может также потребоваться определение с помощью обучения модели среды. В разделе 21.3 рассматривается **активное обучение**, в ходе которого агент должен также определить, что следует делать. Принципиальной проблемой является **исследование** среды: агент должен проводить в своей среде максимально возможное количество экспериментов, для того чтобы определить, как следует в ней действовать. В разделе 21.4 показано, что агент может использовать индуктивное обучение, чтобы как можно быстрее обучиться на своем опыте. В разделе 21.5 рассматриваются методы определения с помощью обучения непосредственных представлений стратегий в рефлексных агентах. Для освоения материала данной главы крайне важно понимание тематики марковских процессов принятия решений (см. главу 17).

## 21.2. ПАССИВНОЕ ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

Для того чтобы упростить изложение, начнем с описания случая пассивного обучающегося агента, в котором используется представление на основе состояний в полностью наблюдаемой среде. При пассивном обучении стратегия агента  $\pi$  является неизменной; это означает, что в состоянии  $s$  он всегда выполняет действие  $\pi(s)$ . Цель агента состоит в том, чтобы определить с помощью обучения, насколько успешной является эта стратегия, т.е. определить с помощью обучения функцию полезности  $U^\pi(s)$ . В этом разделе в качестве примера будет использоваться мир  $4 \times 3$ , представленный в главе 17. На рис. 21.1 для этого мира показаны стратегия и соответствующие полезности. Очевидно, что задача пассивного обучения аналогична задаче **оценки стратегии**, которая является частью алгоритма **итерации по стратегиям**, описанного в разделе 17.3. Основное различие состоит в том, что пассивный обу-

чающийся агент не знает **модели перехода**  $T(s, a, s')$ , которая определяет вероятность достижения состояния  $s'$  из состояния  $s$  после выполнения действия  $a$ ; он также не знает **функцию вознаграждения**  $R(s)$ , которая задает вознаграждение для каждого состояния.

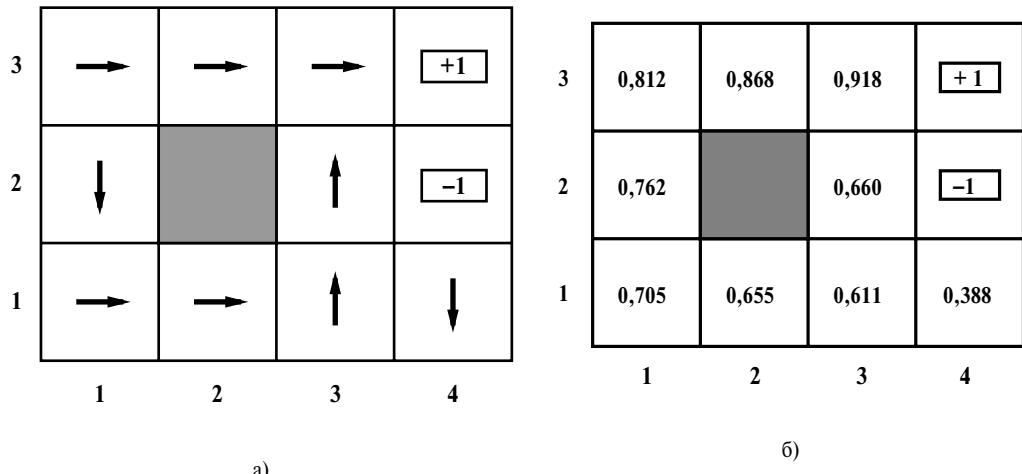


Рис. 21.1. Описание рассматриваемой среды: а) стратегия  $\pi$  для мира  $4 \times 3$ ; оказалось, что эта стратегия является оптимальной при вознаграждениях  $R(s) = -0.04$  в нетерминальных состояниях и при отсутствии обесценивания (а); полезности состояний в мире  $4 \times 3$  с учетом стратегии  $\pi$  (б)

Агент выполняет в данной среде ряд **попыток**, используя свою стратегию  $\pi$ . При осуществлении каждой попытки агент начинает с состояния  $(1, 1)$  и испытывает некоторую последовательность переходов между состояниями до тех пор, пока не достигнет одного из терминальных состояний,  $(4, 2)$  или  $(4, 3)$ . В результатах восприятий ему сообщается и текущее состояние, и вознаграждение, полученное в этом состоянии. Типичные попытки могут выглядеть примерно так:

$$(1, 1) \xrightarrow{-.04} (1, 2) \xrightarrow{-.04} (1, 3) \xrightarrow{-.04} (1, 2) \xrightarrow{-.04} (1, 3) \xrightarrow{-.04} (2, 3) \xrightarrow{-.04} (3, 3) \xrightarrow{-.04} (4, 3) +1$$

$$(1, 1) \xrightarrow{-.04} (1, 2) \xrightarrow{-.04} (1, 3) \xrightarrow{-.04} (2, 3) \xrightarrow{-.04} (3, 3) \xrightarrow{-.04} (3, 2) \xrightarrow{-.04} (3, 3) \xrightarrow{-.04} (4, 3) +1$$

$$(1, 1) \xrightarrow{-.04} (2, 1) \xrightarrow{-.04} (3, 1) \xrightarrow{-.04} (3, 2) \xrightarrow{-.04} (4, 2) -1$$

Обратите внимание на то, что результаты восприятия каждого состояния сопровождаются нижним индексом с указанием полученного вознаграждения. Цель состоит в том, чтобы использовать эту информацию о вознаграждении для определения с помощью обучения ожидаемой полезности  $U^\pi(s)$ , связанной с каждым нетерминальным состоянием  $s$ . Определяемая полезность должна представлять собой ожидаемую сумму (обесцениваемых) вознаграждений, полученных, если агент придерживается стратегии  $\pi$ . Как и в уравнении 17.3, это соотношение записывается следующим образом:

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0=s \right] \quad (21.1)$$

Мы будем включать **коэффициент обесценивания**  $\gamma$  во все уравнения, приведенные в данной главе, но для мира  $4 \times 3$  принято использовать значение  $\gamma=1$ .

### Непосредственная оценка полезности

Простой метод **непосредственной оценки полезности** был изобретен в конце 1950-х годов в области **адаптивной теории управления** Видроу и Хоффом [1587]. Идея этого метода состоит в том, что полезностью данного конкретного состояния является ожидаемое суммарное вознаграждение, связанное с действиями, выполняемыми, начиная с этого состояния, а каждая попытка представляет собой выборку этого значения для каждого посещенного состояния. Например, первая попытка из трех приведенных выше предоставляет одну выборку с суммарным вознаграждением 0.72 для состояния  $(1, 1)$ , две выборки со значениями 0.76 и 0.84 для состояния  $(1, 2)$ , две выборки со значениями 0.80 и 0.88 для состояния  $(1, 3)$  и т.д. Таким образом, в конце каждой последовательности алгоритм вычисляет наблюдаемое будущее вознаграждение для каждого состояния и обновляет соответствующим образом оценку полезности для этого состояния путем ведения текущего среднего значения для каждого состояния в таблице. В пределе, после выполнения бесконечного количества попыток, среднее по выборкам сходится к значению истинного ожидания, приведенному в уравнении 21.1.

Очевидно, что непосредственная оценка полезности представляет собой один из видов контролируемого обучения, в котором каждый пример задает состояние в качестве входных данных, а наблюдаемое будущее вознаграждение — в качестве выходных. Это означает, что данный метод позволяет свести обучение с подкреплением к стандартной задаче индуктивного обучения, которая рассматривалась в главе 18. В разделе 21.4 описано использование более мощных видов представлений для функции полезности, таких как нейронные сети. Методы обучения для этих представлений могут применяться непосредственно к наблюдаемым данным.

Метод непосредственной оценки полезности позволяет успешно свести задачу обучения с подкреплением к задаче индуктивного обучения, о которой уже многое известно. Но, к сожалению, этот метод не позволяет воспользоваться очень важным источником информации — в нем не учитывается тот факт, что полезности состояний не являются независимыми! Дело в том, что *полезность каждого состояния равна сумме его собственного вознаграждения и ожидаемой полезности его состояний-преемников*. Это означает, что значения полезности подчиняются уравнениям Беллмана для данной конкретной стратегии (см. также уравнение 17.10):

$$U^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^\pi(s') \quad (21.2)$$

Поскольку в методе непосредственной оценки полезности игнорируются связи между состояниями, он не позволяет воспользоваться дополнительными возможностями для обучения. Например, во второй из трех попыток, приведенных выше, достигается состояние  $(3, 2)$ , которое еще не было до сих пор посещено агентом. После следующего перехода агент достигает состояния  $(3, 3)$ , которое, как известно из первой попытки, имеет высокую полезность. Уравнение Беллмана позволяет сразу же определить, что состояние  $(3, 2)$  также, по-видимому, будет иметь высокую по-

лезнность, поскольку оно ведет к состоянию  $(3, 3)$ , но метод непосредственной оценки полезности не позволяет ничего определить с помощью обучения до конца этой попытки. В более широком контексте метод непосредственной оценки полезности можно рассматривать как поиск в пространстве гипотез для  $U$ , которое имеет размеры намного большие, чем необходимо, поскольку включает также много функций, которые нарушают уравнения Беллмана. По этой причине данный алгоритм часто сходится очень медленно.

### Адаптивное динамическое программирование

Для того чтобы можно было воспользоваться преимуществами наличия информации об ограничениях между состояниями, агент должен определить с помощью обучения, как связаны эти состояния. Агент, действующий по принципу **адаптивного динамического программирования** (или сокращенно **ADP** — Adaptive Dynamic Programming), функционирует путем определения с помощью обучения модели перехода в этой среде по мере выполнения своих действий и находит решение в соответствующем марковском процессе принятия решений, используя метод динамического программирования. Для пассивного обучающегося агента такой подход означает, что он должен подставлять полученную с помощью обучения модель перехода  $T(s, \pi(s), s')$  и наблюдаемые вознаграждения  $R(s)$  в уравнения Беллмана 21.2 для вычисления полезностей состояний. Как было отмечено при обсуждении в главе 17 принципа итерации по стратегиям, эти уравнения являются линейными (для них не требуется максимизация), поэтому они могут быть решены с помощью любого пакета линейной алгебры. Еще один вариант состоит в том, что может быть принят подход, основанный на принципе **модифицированной итерации по стратегиям** (см. с. 831), в котором используется упрощенный процесс итерации по значениям для обновления оценок полезностей после каждого изменения в модели, определяемой с помощью обучения. Поскольку эта модель после каждого наблюдения подвергается только незначительным изменениям, в процессе итерации по значениям в качестве начальных значений могут использоваться предыдущие оценки полезностей, а сами вычисления с помощью этого метода должны сходиться очень быстро.

Сам процесс определения модели с помощью обучения осуществляется просто, поскольку среда полностью наблюдаема. Это означает, что данная задача обучения представляет собой задачу контролируемого обучения, в которой входными данными являются пары “состояние–действие”, а выходными — результирующие состояния. В простейшем случае модель перехода можно представить как таблицу вероятностей и следить за тем, насколько часто возникает каждый результат действия<sup>1</sup>, оценивая вероятность перехода  $T(s, a, s')$  по данным о частоте, с которой достигается состояние  $s'$  при выполнении действия  $a$  в состоянии  $s$ . Например, в трех трассировках попыток, приведенных на с. 1013, действие *Right* в состоянии  $(1, 3)$  выполняется три раза, и двумя из трех результирующих состояний становится состояние  $(2, 3)$ , поэтому вероятность перехода  $T((1, 3), Right, (2, 3))$  оценивается как  $2/3$ .

<sup>1</sup> Это — оценка максимального правдоподобия, которая рассматривается в главе 20. Способ байесовского обновления с использованием распределения априорных вероятностей Дирихле может оказаться лучше.

Полная программа агента для пассивного агента ADP приведена в листинге 21.1. Производительность этой программы в мире  $4 \times 3$  показана на рис. 21.2. Судя по тому, насколько быстро улучшаются полученные им оценки значений, агент ADP действует настолько успешно, насколько это возможно, учитывая его способность определять с помощью обучения модель перехода. В этом смысле данный проект агента может служить эталоном, с которым можно будет сравнивать производительность других алгоритмов обучения с подкреплением. Тем не менее в больших пространствах состояний такой подход становится не вполне осуществимым. Например, в нарядах для его использования потребуется решить приблизительно  $10^{50}$  уравнений с  $10^{50}$  неизвестными.

**Листинг 21.1. Алгоритм агента для пассивного обучения с подкреплением, основанный на аддитивном динамическом программировании. Для упрощения кода было принято предположение, что каждый результат восприятия можно разделить на информацию о состоянии и сигнал о вознаграждении**

---

```

function Passive-ADP-Agent(percept) returns действие a
    inputs: percept, результаты восприятия, обозначающие текущее
             состояние s' и сигнал вознаграждения r'
    static:  $\pi$ , зафиксированная стратегия
             mdp, марковский процесс принятия решений с моделью T,
             вознаграждениями R, коэффициентом обесценивания  $\gamma$ 
             U, таблица значений полезностей, первоначально пустая
             Nsa, таблица частот пар "состояние-действие", первоначально
             содержащая нули
             Nsas', таблица частот троек "состояние-действие-состояние",
             первоначально содержащая нули
             s, a, предыдущие состояние и действие, первоначально пустые

    if состояние s' является новым then do
        U[s']  $\leftarrow r'$ ; R[s']  $\leftarrow r'$ 
    if состояние s не пусто then do
        увеличить значения Nsa[s, a] и Nsas'[s, a, s']
        for each t, такого что значение Nsas'[s, a, t] является
        ненулевым do
            T[s, a, t]  $\leftarrow N_{sas'}[s, a, t] / N_{sa}[s, a]
    U  $\leftarrow$  Value-Determination( $\pi$ , U, mdp)
    if Terminal?[s'] then s, a  $\leftarrow$  пустые значения else s, a  $\leftarrow s', \pi[s']
    return a$$ 
```

---

## Обучение с учетом временной разницы

Существует также возможность взять (почти) самое лучшее из обоих описанных выше подходов; это означает, что можно аппроксимировать приведенные выше уравнения ограничений, не решая их для всех возможных состояний. Ключом к созданию этого метода становится то, что можно использовать наблюдаемые переходы для корректировки значений наблюдаемых состояний так, чтобы они согласовывались с уравнениями ограничений. Рассмотрим, например, переход из состояния  $(1, 3)$  в состояние  $(2, 3)$  во второй попытке, показанной на с. 1013. Предположим, что в результате первой попытки были получены оценки полезности  $U^t(1, 3) = 0.84$

и  $U^\pi(2, 3) = 0.92$ . Итак, если будет постоянно происходить этот переход, то следует учитывать, что указанные полезности будут подчиняться следующему уравнению:

$$U^\pi(1, 3) = -0.04 + U^\pi(2, 3)$$

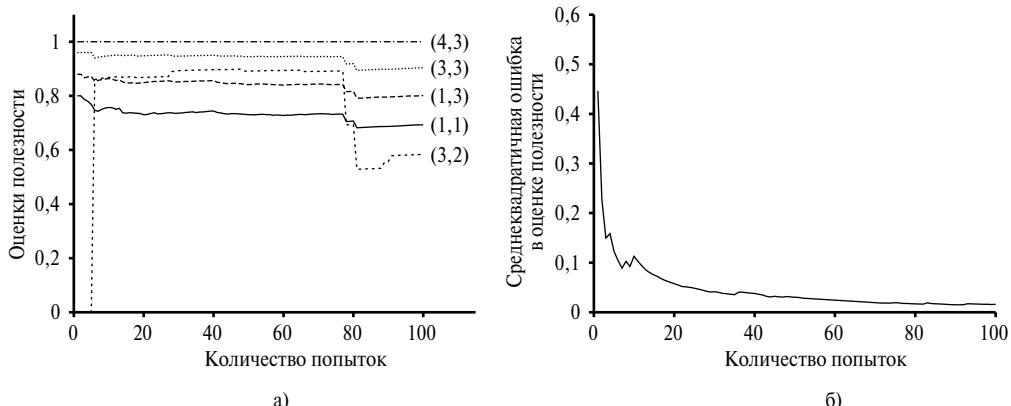


Рис. 21.2. Кривые пассивного обучения ADP для мира  $4 \times 3$ , полученные при оптимальной стратегии, которая показана на рис. 21.1: оценки полезности для избранного подмножества состояний, полученные как функции от количества попыток (а). Обратите внимание на то, что примерно при 78-й попытке происходит значительные изменения; именно тогда агент впервые попадает в терминальное состояние с полезностью  $-1$ , соответствующее квадрату  $(4, 2)$ ; среднеквадратичная ошибка в оценке для  $U(1, 1)$ , усредненная по 20 прогонам, состоящим из 100 попыток каждый (б)

поэтому значение  $U^\pi(1, 3)$  будет равно 0.88. Таким образом, текущая оценка этой полезности, равная 0.84, может оказаться немного заниженной и должна быть увеличена. Более общий вывод состоит в том, что если происходит переход из состояния  $s$  в состояние  $s'$ , то к значению полезности  $U^\pi(s)$  применяется следующее обновление:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s)) \quad (21.3)$$

где  $\alpha$  — параметр **скорости обучения**. Поскольку в этом правиле обновления используется разность между полезностями последовательных состояний, соответствующее уравнение часто называют уравнением **временной разности**, или сокращенно **TD** (Temporal Difference).

Основная идея всех методов временной разности состоит в том, что вначале определяются условия, выполняемые локально, когда оценки полезностей являются правильными, а затем составляется уравнение обновления, в котором оценки переносятся в это идеальное уравнение “равновесия”. В случае пассивного обучения равновесие задается уравнением 21.2. Теперь уравнение 21.3 по сути вынуждает агента достичь равновесия, заданного в уравнении 21.2, но с учетом некоторых нюансов. Прежде всего следует отметить, что данное обновление касается только наблюдаемого преемника  $s'$ , тогда как фактические условия равновесия касаются всех возможных следующих состояний. Можно было бы предположить, что это вызовет необоснованно большие изменения в значении  $U^\pi(s)$  при возникновении очень редких переходов, но фактически, поскольку эти редкие переходы действительно случаются крайне редко, среднее значение  $U^\pi(s)$  сходится к правильному значению.

Более того, если в качестве коэффициента  $\alpha$  вместо фиксированного параметра будет применяться функция со значением, уменьшающимся по мере увеличения количества случаев посещения некоторого состояния, то само значение  $U(s)$  будет сходиться к правильному значению<sup>2</sup>. Эти рассуждения позволяют составить программу агента, приведенную в листинге 21.2. На рис. 21.3 показана производительность пассивного агента TD в мире  $4 \times 3$ . Обучение этого агента происходит менее быстро по сравнению с агентом ADP, и он показывает более значительную изменчивость, но сам алгоритм агента гораздо проще и требует меньше вычислений в расчете на каждое наблюдение. Обратите внимание на то, что ~~о~~ алгоритм TD не требует применения модели для осуществления предусмотренных в нем обновлений. Информацию о связях между соседними состояниями поставляет сама среда в форме наблюдаемых переходов.

**Листинг 21.2. Алгоритм агента для пассивного обучения с подкреплением, который позволяет определить с помощью обучения оценки полезностей на основе временных разностей**

---

```

function Passive-TD-Agent(percept) returns действие a
    inputs: percept, результаты восприятия, обозначающие текущее
             состояние s' и сигнал вознаграждения r'
    static:  $\pi$ , зафиксированная стратегия
             U, таблица значений полезностей, первоначально пустая
             Ns, таблица частот состояний, первоначально содержащая нули
             s, a, r, предыдущее состояние, действие и вознаграждение,
             первоначально пустые

    if состояние s' является новым then  $U[s'] \leftarrow r'$ 
    if состояние s не пусто then do
        увеличить значение Ns[s]
         $U[s] \leftarrow U[s] + \alpha(\{N_s[s]\})(r + \gamma U[s'] - U[s])$ 
    if Terminal?[s'] then s, a, r  $\leftarrow$  пустые значения
    else s, a, r  $\leftarrow s', \pi[s'], r'$ 
    return a

```

---

Подход на основе ADP и подход на основе TD фактически тесно связаны. В обоих этих алгоритмах предпринимаются попытки внести локальные корректировки в оценки полезностей, для того чтобы обеспечить “согласование” каждого состояния с его преемниками. Одно из различий между ними состоит в том, что в алгоритме TD состояние корректируется для согласования с его наблюдаемым преемником (уравнение 21.3), а в алгоритме ADP состояние корректируется для согласования со всеми преемниками, которые могут быть получены с учетом весов их вероятностей (уравнение 21.2). Это различие исчезает, когда результаты корректировок TD усредняются по большому количеству переходов, поскольку частота появления каждого преемника в множестве переходов приблизительно пропорциональна его вероятности. Более важное различие состоит в том, что в алгоритме TD выполняется отдельная корректировка в расчете на каждый наблюдаемый переход, а в алгоритме ADP выполняется столько корректировок, сколько требуется для восстановления согла-

---

<sup>2</sup> Формально требуется, чтобы соблюдались условия  $\sum_{n=1}^{\infty} \alpha(n) = \infty$  и  $\sum_{n=1}^{\infty} \alpha^2(n) < \infty$ . Этим условиям удовлетворяет функция затухания  $\alpha(n)=1/n$ , а на рис. 21.3 использовалась функция  $\alpha(n)=60/(59+n)$ .

сованности между оценками полезностей  $U$  и моделью среды  $T$ . Хотя наблюдаемый переход вносит в  $T$  только локальное изменение, его результаты могут потребовать распространения по всем полезностям  $U$ . Таким образом, алгоритм TD может рассматриваться как грубое, но эффективное первое приближение алгоритма ADP.

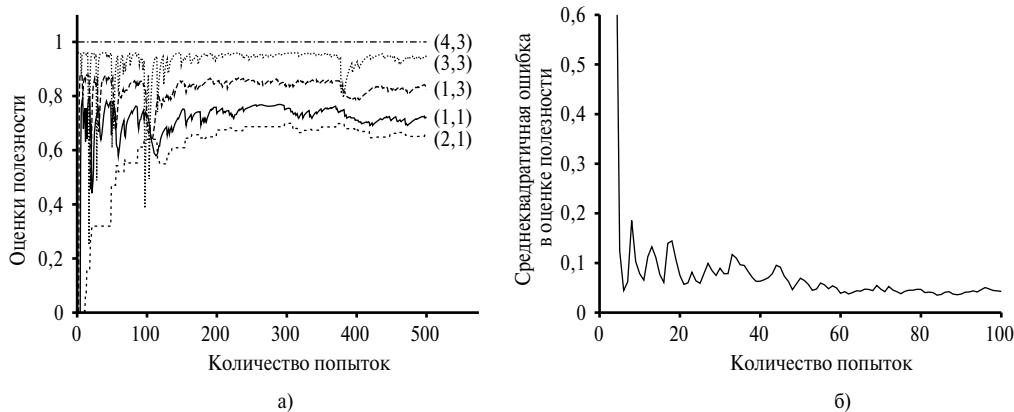


Рис. 21.3. Кривые обучения с помощью алгоритма TD для мира  $4 \times 3$ : оценки полезности для избранного подмножества состояний, полученные как функции от количества попыток (а); среднеквадратичная ошибка в оценке для  $U(1,1)$ , усредненная по 20 прогонам, состоящим из 500 попыток каждый (б). Показаны результаты только для первых 100 попыток, чтобы можно было провести сравнение с рис. 21.2

Каждая корректировка, внесенная алгоритмом ADP, с точки зрения пользователя алгоритма TD может рассматриваться как результат “псевдоэксперимента”, полученный путем моделирования в текущей модели среды. Подход, предусмотренный в алгоритме TD, можно дополнить в целях использования модели среды для выработки результатов нескольких псевдоэкспериментов, иначе говоря, переходов, возможность осуществления которых агент TD может допустить согласно его текущей модели. В ответ на каждый наблюдаемый переход агент TD может вырабатывать большое количество воображаемых переходов. Благодаря этому результирующие оценки полезностей TD будут все больше и больше аппроксимировать соответствующие оценки ADP, разумеется, за счет увеличения продолжительности времени вычислений.

Аналогичным образом могут создаваться все более эффективные версии алгоритма ADP путем непосредственной аппроксимации алгоритмов для итерации по значениям или итерации по стратегиям. Напомним, что полная итерация по значениям может быть трудновыполнимой, когда количество состояний велико. Однако многие этапы корректировки являются чрезвычайно малыми. Одним из возможных подходов, применимых для быстрой выработки достаточно качественных ответов, является ограничение количества корректировок, внесенных после каждого наблюдаемого перехода. Можно также использовать какую-то эвристическую для ранжирования возможных корректировок, с тем чтобы в дальнейшем осуществлять только наиболее значимые из них. Эвристика, предусматривающая **выметание с учетом приоритетов**, позволяет предпочесть вариант с внесением корректировок в состояния, возможные преемники которых уже подвергались большими корректировкам в их собственных оценках полезностей. Приближенные алгоритмы ADP, в которых

используются подобные эвристики, обычно обеспечивают обучение примерно с таким же быстродействием, как и полные алгоритмы ADP, с точки зрения количества обучающих последовательностей, но могут оказаться на несколько порядков величины более эффективными с точки зрения объема вычислений (см. упр. 21.3.) Это дает возможность применять такие алгоритмы для обработки пространств состояний, намного превышающих по размерам те, которые являются приемлемыми для полного алгоритма ADP. Приближенные алгоритмы ADP имеют еще одно дополнительное преимущество: на ранних этапах обучения в новой среде модель среды  $T$  часто далека от правильной, поэтому нет смысла слишком точно вычислять функцию полезности для согласования с ней. Приближенный алгоритм позволяет использовать минимальную величину корректировки, которая уменьшается по мере того, как модель среды становится все более точной. Это позволяет устранить очень продолжительные итерации по значениям, которые могут возникать на ранних этапах обучения из-за больших изменений в модели.

### 21.3. АКТИВНОЕ ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

Пассивный обучающийся агент руководствуется постоянно заданной стратегией, которая определяет его поведение, а активный агент должен сам принимать решение о том, какие действия следует предпринять. Начнем с описания агента, действующего с помощью адаптивного динамического программирования, и рассмотрим, какие изменения необходимо внести в его проект, чтобы он мог функционировать с учетом этой новой степени свободы.

Прежде всего агенту потребуется определить с помощью обучения полную модель с вероятностями результатов для всех действий, а не просто модель для заданной стратегии. Для этой цели превосходно подходит простой механизм обучения, используемый в алгоритме Passive-ADP-Agent. Затем необходимо принять в расчет тот факт, что агент должен осуществлять выбор из целого ряда действий. Полезности, которые ему потребуются для обучения, определяются оптимальной стратегией; они подчиняются уравнениям Беллмана, приведенным на с. 824, которые мы еще раз приведем ниже для удобства.

$$U(s) = R(s) + \gamma \max_s \sum_{s'} T(s, a, s') U(s') \quad (21.4)$$

Эти уравнения могут быть решены для получения функции полезности  $U$  с помощью алгоритмов итерации по значениям или итерации по стратегиям, приведенных в главе 17. Последняя задача состоит в определении того, что делать на каждом этапе. Получив функцию полезности  $U$ , оптимальную для модели, определяемой с помощью обучения, агент может извлечь информацию об оптимальном действии, составляя одношаговый прогноз для максимизации ожидаемой полезности; еще один вариант состоит в том, что если используется итерация по стратегиям, то оптимальная стратегия уже известна, поэтому агент должен просто выполнить действие, рекомендуемое согласно оптимальной стратегии. Но действительно ли он должен выполнять именно это действие?

## Исследование среды

На рис. 21.4 показаны результаты одной последовательности попыток для агента ADP, который следует рекомендациям по выбору оптимальной стратегии для модели, определяемой с помощью обучения, на каждом этапе. Как оказалось, агент не находит с помощью обучения истинные полезности или истинную оптимальную стратегию! Вместо этого происходит то, что после 39-й попытки агент находит стратегию, позволяющую достичь вознаграждения +1 вдоль нижнего маршрута, проходящего через квадраты  $(2, 1)$ ,  $(3, 1)$ ,  $(3, 2)$  и  $(3, 3)$  (рис. 21.4). После проведения экспериментов с небольшими вариантами, начиная от 276-й попытки и дальше, агент постоянно придерживается этой стратегии, так и не определив с помощью обучения полезности других состояний и не найдя оптимальный маршрут через квадраты  $(1, 2)$ ,  $(1, 3)$  и  $(2, 3)$ . Авторы называют такого агента, действующего с помощью жадного алгоритма, просто **жадным агентом**. Повторные эксперименты показали, что поиски жадного агента очень редко сходятся в пределе к оптимальной стратегии для данной среды, а иногда сходятся к таким стратегиям, которые являются действительно устрашающими по своей неэффективности.

Как могло оказаться, что выбор оптимального действия приводит к неоптимальным результатам? Ответ состоит в том, что модель, определяемая с помощью обучения, не является такой же, как истинная среда; поэтому то, что оптимально в модели, определяемой с помощью обучения, может оказаться неоптимальным в истинной среде. К сожалению, агент не имеет информации о том, какова истинная среда, поэтому не может вычислить оптимальное действие для истинной среды. Так что же делать?

В проекте жадного агента не учтено то, что действия не только предоставляют вознаграждения в соответствии с моделью, определяемой в настоящее время с помощью обучения, но и вносят вклад в определение с помощью обучения самой истинной модели, влияя на полученные результаты восприятия. Совершенствуя эту модель, агент сможет получать большие вознаграждения не сразу же, а в будущем<sup>3</sup>. Поэтому агент должен искать компромисс между **потреблением** полученных результатов для максимизации своего вознаграждения (что отражается в его текущих оценках полезностей) и **исследованием** среды для максимизации своего долговременного благосостояния. Занимаясь исключительно потреблением полученных благ, агент рискует застрять в одной колее, а занимаясь исключительно исследованием для повышения уровня своих знаний, агент не получит пользы, если так и не внедрит эти знания на практике. В реальном мире человеку постоянно приходится решать, стоит ли продолжать беззаботное существование или нужно окунуться в неизвестность в надежде найти новые и лучшие условия жизни. Но чем большими знаниями он обладает, тем меньше нуждается в дальнейших исследованиях.

Можно ли выработать более точные рекомендации по сравнению с этими общими рассуждениями? Существует ли оптимальный способ организации исследования среды? Как оказалось, эти вопросы глубоко изучались в той области статистической теории принятий решений, которая касается так называемых **задач с n-рукими бандитами**, — так принято называть игорные автоматы, управляемые с помощью рукояток (см. врезку).

<sup>3</sup> Обратите внимание на то, что здесь просматривается прямая аналогия с теорией ценности информации, описанной в главе 16.

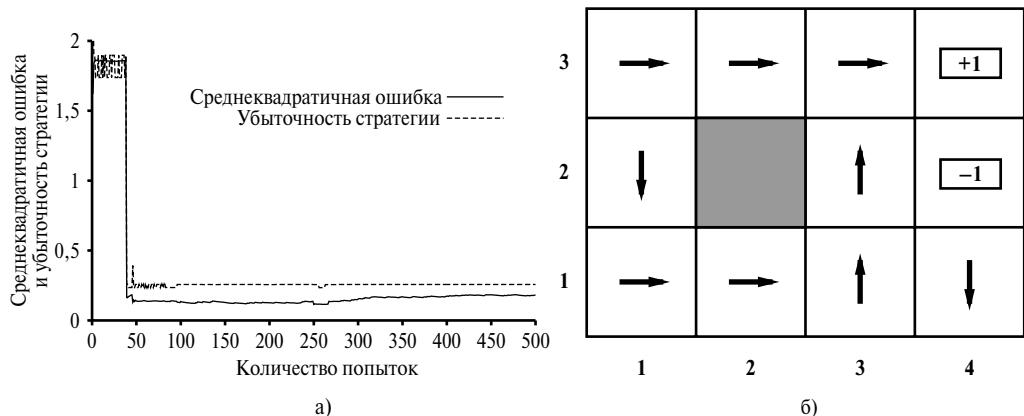


Рис. 21.4. Производительность агента *ADP*, действующего с помощью жадного алгоритма, который осуществляет действие, рекомендованное согласно оптимальной стратегии для модели, определяемой с помощью обучения: среднеквадратичная ошибка в оценках полезностей, усредненная по девяти нетерминальным квадратам (а); неоптимальная стратегия, к которой в пределе сходится поиск стратегии, выполняемый агентом с помощью жадного алгоритма, в данной конкретной последовательности попыток (б)

### ИССЛЕДОВАНИЕ СРЕДЫ И N-РУКИЕ БАНДИТЫ

В Лас-Вегасе одноруким бандитом называют игорный автомат определенного типа, в который игрок может вложить монету, потянуть за рукоятку и забрать выигрыш (если только таковой действительно появится). Существует также разновидность этого автомата с  $n$  рукоятками, называемая  **$n$ -руким бандитом**. Игрок должен выбрать, какую рукоятку следует потянуть на себя после вкладывания каждой следующей монеты, — ту, которая когда-то дала наибольший выигрыш, а может быть, ту, которую он еще не пытался использовать?

Задача с  $n$ -руким бандитом — это формальная модель реальных задач во многих жизненно важных областях, таких как принятие решения о выделении годового бюджета на исследования и разработки по искусственному интеллекту. Каждая рукоятка соответствует определенному действию (такому как выделение 20 миллионов долларов на подготовку новых учебников по искусственному интеллекту), а вознаграждение за подтягивание к себе такой рукоятки соответствует прибыли, полученной от выполнения соответствующего действия (в данном случае — просто колоссальной). Исследование среды, будь то исследование перспектив нового научного направления или зондирование нового товарного рынка, является рискованным, дорогостоящим и связано с неопределенными вознаграждениями; с другой стороны, полный отказ от проведения исследований означает, что не удастся обнаружить новые сферы деятельности, которые могут оказаться прибыльными.

Чтобы правильно сформулировать задачу с  $n$ -руким бандитом, необходимо точно определить, что подразумевается под оптимальным поведением. Большинство определений, приведенных в литературе, основано на предположении, что цель состоит в максимизации ожидаемого суммарного вознаграждения, полученного в течение всего срока существования агента. В этих опреде-

лениях требуется, чтобы ожидаемое вознаграждение оценивалось по всем возможным мирам, в которых может оказаться агент, а также по возможным результатам каждой последовательности действий в любом конкретном мире. В данном случае “мир” определяется моделью перехода  $T(s, a, s')$ . Таким образом, для того чтобы действовать оптимальным образом, агент должен знать распределение априорных вероятностей по всем возможным моделям. Возникающие в конечном итоге задачи оптимизации обычно являются крайне трудно разрешимыми.

В некоторых случаях (например, когда вознаграждение, получаемое от игры на каждом автомате, является независимым и используются обесцениваемые вознаграждения) существует возможность рассчитать **индекс Гиттинса** для каждого игорного автомата [561]. Этот индекс представляет собой функцию, параметрами которой являются только количество игр, проведенных на игорном автомате, и сумма полученного выигрыша. Индекс для каждого автомата показывает, насколько оправданы дополнительные затраты денег на продолжение игры с учетом комбинации ожидаемой прибыли и ожидаемой стоимости информации. Выбор автомата с наибольшим значением индекса позволяет найти оптимальную стратегию исследования среды. К сожалению, до сих пор не было обнаружено ни одного метода, позволяющего распространить понятие индексов Гиттинса на проблематику задач последовательного принятия решений.

Теорию  $n$ -руких бандитов можно использовать для обоснования приемлемости стратегии отбора в генетических алгоритмах (см. главу 4). Если в задаче с  $n$ -руким бандитом каждая рукоятка рассматривается как возможная строка генов, а вкладывание монеты для подтягивания одной рукоятки — как воспроизведение этих генов, то генетические алгоритмы должны обеспечить оптимальное вложение монет в игорный автомат с учетом соответствующего множества предположений о независимости.

Хотя задачи с  $n$ -рукими бандитами чрезвычайно трудно поддаются точному решению для получения оптимального метода исследования среды, тем не менее возможно предложить приемлемую схему, которая в конечном итоге приводит к оптимальному поведению со стороны агента. Формально любая такая схема должна быть жадной в пределе бесконечного исследования; это свойство сокращенно обозначается как  $\bowtie$  GLIE (Greedy in the Limit of Infinite Exploration). Схема GLIE должна предусматривать опробование каждого действия в каждом состоянии путем осуществления неограниченного количества попыток такого действия для предотвращения появления конечной вероятности того, что будет пропущено какое-то оптимальное действие из-за крайне неудачного стечения обстоятельств. Агент ADP, в котором используется такая схема, должен в конечном итоге определить с помощью обучения истинную модель среды. Кроме того, схема GLIE должна в конечном итоге стать жадной, для того чтобы действия агента стали оптимальными по отношению к определенной с помощью обучения (и поэтому истинной) модели.

Предложено несколько схем GLIE; в одной из простейших из этих схем предусматривается, что агент должен в течение  $1/t$  части времени выбирать случайное действие, а в течение остального времени придерживаться жадной стратегии. Хотя такая схема в конечном итоге сходится к оптимальной стратегии, весь этот процесс

может оказаться чрезвычайно замедленным. Более обоснованный подход предусматривает присваивание определенных весов тем действиям, которые агент не опробовал очень часто, стараясь вместе с тем избегать действий, которые считаются имеющими низкую полезность. Такой подход может быть реализован путем модификации уравнения ограничения 21.4 таким образом, чтобы в нем присваивалась более высокая оценка полезности относительно мало исследованным парам “состояние–действие”. По сути это сводится к определению оптимистического распределения априорных вероятностей по возможным вариантам среды и вынуждает агента на первых порах вести себя так, как если бы повсеместно были разбросаны замечательные вознаграждения. Допустим, что для обозначения оптимистической оценки полезности состояния  $s$  (т.е. ожидаемого будущего вознаграждения) используется запись  $U^*(s)$ , и предположим, что  $N(a, s)$  — количество попыток опробования действия  $a$  в состоянии  $s$ . Предположим также, что в обучающемся агенте ADP используется метод итерации по значениям; в таком случае необходимо переписать уравнение обновления (т.е. уравнение 17.6), чтобы включить в него оптимистическую оценку. Именно такая цель достигается с помощью следующего уравнения:

$$U^*(s) \leftarrow R(s) + \gamma \max_a f \left( \sum_{s'} T(s, a, s') U^*(s'), N(a, s) \right) \quad (21.5)$$

где  $f(u, n)$  называется **функцией исследования**. Эта функция определяет компромисс между жадностью (предпочтениями, отложенными высоким значениям  $u$ ) и любопытством (предпочтениями, отложенными низким значениям  $n$ , характеризующим действия, которые еще не были опробованы достаточно часто). Функция  $f(u, n)$  должна увеличиваться в зависимости от  $u$  и уменьшаться в зависимости от  $n$ . Безусловно, что существует много возможных функций, которые соответствуют этим условиям. Одним из особенно простых определений такой функции является следующее:

$$f(u, n) = \begin{cases} R^+ & \text{если } n < N_e \\ u & \text{в противном случае} \end{cases}$$

где  $R^+$  — оптимистическая оценка наилучшего возможного вознаграждения, которое может быть получено в любом состоянии;  $N_e$  — постоянный параметр. Результатом применения такой функции становится то, что агент пытается проверить каждую пару “состояние–действие” по мере  $N_e$  раз.

Тот факт, что в правой части уравнения 21.5 присутствует величина  $U^*$ , а не  $U$ , очень важен. Может вполне оказаться так, что по мере развития процесса исследования в большом количестве попыток будут опробованы состояния и действия, находящиеся недалеко от начального состояния. Если бы использовалась более пессимистическая оценка полезности,  $U$ , то агент вскоре стал бы не склонным проводить дальнейшее исследование среды. А применение оценки  $U^*$  означает, что выгоды от исследования среды распространяются в обратном направлении от границ неисследованных регионов, поэтому получают больший вес действия, ведущие к неисследованным регионам, а не просто действия, которые сами по себе остаются малоизученными. Эффект использования такой исследовательской стратегии наглядно показан на рис. 21.5, который демонстрирует более быструю сходимость к оптимальной производительности в отличие от жадного подхода. Способ действий, очень близкий к оптималь-

ному, обнаруживается всего лишь после 18 попыток. Обратите внимание на то, что сами оценки полезности не сходятся так же быстро. Это связано с тем, что агент довольно рано прекращает исследование частей пространства состояний, не предоставляющих вознаграждения, и в дальнейшем посещает их только “по случаю”. Но благодаря такому подходу агент приобретает идеальное понимание того, что не следует задумываться о точных значениях полезностей состояний, которые, как ему известно, являются нежелательными и которых можно избежать.

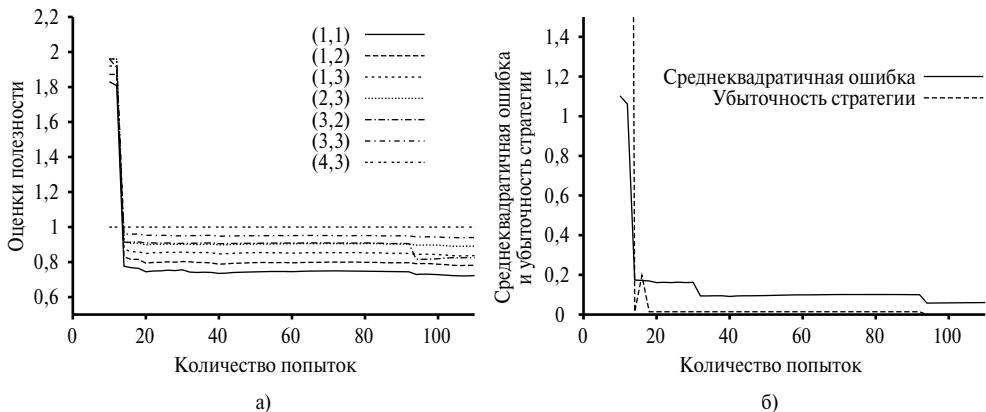


Рис. 21.5. Производительность агента ADP, проводящего исследование среды с использованием параметров  $R^+=2$  и  $N_e=5$ : оценки изменения полезностей для выбранных состояний во времени (а); среднеквадратичная ошибка в значениях полезностей и связанная с ней убыточность стратегии (б)

### Определение функции “действие–стоимость” с помощью обучения

Теперь, после разработки алгоритма активного агента ADP, рассмотрим, как сконструировать активного агента, обучающегося по методу временной разности. Наиболее очевидным отличием от пассивного варианта является то, что агенту больше не предоставлена заданная стратегия, поэтому для определения с помощью обучения функции полезности  $U$  агенту потребуется определить с помощью обучения некоторую модель, чтобы иметь возможность выбрать с учетом значения  $U$  некоторое действие, выполнив прогнозирование на один шаг вперед. Задача приобретения модели для агента TD идентична такой же задаче для агента ADP. А что можно сказать о самом правиле обновления TD? Возможно, что следующее утверждение на первый взгляд покажется удивительным, но правило обновления 21.3 остается неизменным. Такая ситуация может показаться странной по следующей причине: предположим, что агент делает шаг, который обычно приводит в приемлемое место назначения, но из-за наличия недетерминированности в самой среде в конечном итоге агент оказывается в катастрофическом состоянии. Правило обновления TD позволяет отнести к этой ситуации так же серьезно, как если бы полученным итогом был нормальный результат действия, тогда как можно было предположить, что агент не должен слишком сильно об этом беспокоиться, поскольку такой итог возник лишь из-за стечения обстоятельств. Безусловно, такой маловероятный результат в большом множестве обучающих последовательностей может возникать

достаточно редко, поэтому можно надеяться, что в долговременной перспективе его последствия получат вес, пропорциональный их вероятности. Еще раз отметим следующее — можно доказать, что алгоритм TD сходится к тем же значениям, что и ADP, по мере того как количество обучающих последовательностей стремится к бесконечности.

Есть также альтернативный метод TD, называемый **Q-обучением**, в котором предусматривается определение с помощью обучения некоторого представления “действие–стоимость” вместо определения полезностей. Для обозначения стоимости выполнения действия  $a$  в состоянии  $s$  будет использоваться запись  $Q(a, s)$ . Отметим, что Q-значения непосредственно связаны со значениями полезностей следующим образом:

$$U(s) = \max_a Q(a, s) \quad (21.6)$$

На первый взгляд может показаться, что Q-функции представляют собой лишь еще один способ хранения информации о полезностях, но они обладают очень важным свойством: *агенту TD, который определяет Q-функцию с помощью обучения, не требуется модель ни для обучения, ни для выбора действия*. По этой причине Q-обучение называют **безмодельным** методом. Как и в случае полезностей, можно записать следующее уравнение для ограничений, которое должно соблюдаться в точке равновесия, когда Q-значения являются правильными:

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s') \quad (21.7)$$

Как и в случае обучающегося агента ADP, это уравнение может непосредственно использоваться в качестве уравнения обновления для процесса итерации, в котором вычисляются точные Q-значения при наличии оцениваемой модели. Но для этого требуется, чтобы с помощью обучения осуществлялось также определение модели, поскольку в уравнении используется вероятность  $T(s, a, s')$ . С другой стороны, в подходе на основе временной разности модель не требуется. Уравнение обновления для Q-обучения по методу TD, которое вычисляется каждый раз, когда в состоянии  $s$ , ведущем к состоянию  $s'$ , выполняется действие  $a$ , является следующим:

$$Q(a, s) \leftarrow Q(a, s) + \alpha(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s)) \quad (21.8)$$

Полный проект агента для исследующего среду Q-обучающегося агента, в котором используется метод TD, приведен в листинге 21.3. Обратите внимание на то, что в нем используется точно такая же функция исследования  $f$ , которая была предусмотрена для исследующего среду агента ADP, поэтому возникает необходимость вести статистические данные о выполненных действиях (таблицу  $N$ ). Если бы применялась более простая исследовательская стратегия (скажем, выбор действий случайным образом в некоторой части этапов, притом что эта часть уменьшается со временем), то можно было бы отказаться от ведения этих статистических данных.

Такой Q-обучающийся агент определяет с помощью обучения оптимальную стратегию для мира  $4 \times 3$ , но достигает этой цели с гораздо меньшей скоростью по сравнению с агентом ADP. Это связано с тем, что метод TD не вынуждает агента добиваться согласованности значений во всей модели. В связи с этим сравнением возникает общий вопрос:

что лучше — определять с помощью обучения модель и функцию полезности или функцию “действие–значение” без модели? Иными словами, в чем состоит наилучший способ представления функции агента? Это — фундаментальный вопрос искусственного интеллекта. Как было указано в главе 1, традиционно одной из ключевых характерных особенностей многих исследований по искусственному интеллекту была (часто не выраженная явно) приверженность подходу,  **основанному на знаниях**. Такой подход сводится к предположению, что наилучший способ задания функции агента состоит в формировании представления некоторых аспектов среды, в которой находится агент.

**Листинг 21.3. Проводящий исследование среды Q-обучающийся агент.** Это — активный ученик, который определяет с помощью обучения значение  $Q(a, s)$  каждого действия в каждой ситуации. В нем используется такая же исследовательская функция  $f$ , как и в проводящем исследование среды агенте ADP, но исключается необходимость определять с помощью обучения модель перехода, поскольку Q-значение любого состояния может быть непосредственно связано с соответствующими значениями его соседних состояний

---

```

function Q-Learning-Agent (percept) returns действие a
    inputs: percept, результаты восприятия, обозначающие текущее
             состояние s' и сигнал вознаграждения r'
    static: Q, таблица значений действий, индексированная
             по состояниям и действиям
             Nsa, таблица частот пар "состояние–действие"
             s, a, r, предыдущие состояние, действие и вознаграждение,
             первоначально пустые

    if состояние s не пусто then do
        увеличить значение Nsa[s, a]
         $Q[a, s] \leftarrow Q[a, s] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[a', s'] - Q[a, s])$ 
    if Terminal?[s'] then s, a, r  $\leftarrow$  пустое значение
    else s, a, r  $\leftarrow$  s', argmax f(Q[a', s'], Nsa[a', s']), r'
    return a

```

---

Некоторые исследователи, и принадлежащие, и не принадлежащие к сообществу специалистов по искусственному интеллекту, выступили с заявлениями, что доступность методов, не требующих применения модели, таких как Q-обучение, означает, что подход, основанный на знаниях, не является необходимым. Тем не менее пока нет почти никаких оснований, позволяющих судить об обоснованности этих заявлений, кроме интуиции. А интуиция авторов в размышлениях о том, какой подход является наиболее перспективным, подсказывает, что по мере усложнения среды преимущества подхода, основанного на знаниях, становятся все более очевидными. Это обнаруживается даже в играх, таких как шахматы, шашки и нарды (см. следующий раздел), где усилия по определению с помощью обучения функции оценки на основе модели увенчались большим успехом, чем методы Q-обучения.

## 21.4. ОБОБЩЕНИЕ В ОБУЧЕНИИ С ПОДКРЕПЛЕНИЕМ

До сих пор в этой главе предполагалось, что функции полезности и Q-функции, определяемые агентами с помощью обучения, представлены в табличной форме

с одним выходным значением для каждого входного кортежа. Такой подход полностью себя оправдывает в небольших пространствах состояний, но время достижения сходимости и (в случае ADP) затраты времени на каждую итерацию быстро возрастают по мере увеличения размеров пространства. При использовании тщательно управляемых, приближенных методов ADP иногда удается справиться с задачами по обработке 10 000 или большего количества состояний. Этого достаточно для двухмерных вариантов среды, подобных лабиринтам, но более реальные миры далеко выходят за эти пределы. Шахматы и нарды представляют собой крошечные подмножества реального мира, но даже их пространства состояний содержат примерно от  $10^{50}$  до  $10^{120}$  состояний. Даже само предположение о том, что нужно было бы посетить все эти состояния, для того чтобы узнать с помощью обучения, как играть в такую игру, является абсурдным!

Один из способов справиться с этими задачами состоит в использовании средств **функциональной аппроксимации**; такая рекомендация просто означает, что для функций следует применять представления любого рода, отличные от таблиц. Такое представление рассматривается как аппроксимированное, поскольку может оказаться, что истинная функция полезности или Q-функция не может быть точно представлена в выбранной форме. Например, в главе 6 была описана **функция оценки** для шахмат, представленная в виде взвешенной линейной функции от множества **характеристик** (или **базисных функций**)  $f_1, \dots, f_n$ :

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \dots + \theta_n f_n(s)$$

Алгоритм обучения с подкреплением позволяет определить с помощью обучения такие значения параметров  $\theta = \theta_1, \dots, \theta_n$ , что функция оценки  $\hat{U}_\theta$  аппроксимирует истинную функцию полезности. Вместо использования, скажем,  $10^{120}$  значений в таблице, такой аппроксиматор функции характеризуется, допустим,  $n=20$  параметрами, а это просто колоссальное сжатие. В частности, хотя никто не знает истинную функцию полезности для шахмат, никто и не считает, что ее можно точно представить с помощью 20 чисел. Но если эта аппроксимация является достаточно качественной, агент все равно приобретает возможность достичь поразительных успехов в шахматах<sup>4</sup>.

Функциональная аппроксимация может дать возможность представить функции полезности для очень больших пространств состояний, но ее основное преимущество состоит не в этом. **Сжатие, достигнутое с помощью аппроксиматора функции, позволяет обучающемуся агенту делать обобщения, распространяющиеся с тех состояний, которые он уже посетил, на состояния, которые он еще не посетил.** Это означает, что наиболее важным аспектом функциональной аппроксимации является не то, что она требует меньше пространства, а то, что она обеспечивает индуктивное обобщение по входным состояниям. Чтобы дать читателю представление о возможностях

---

<sup>4</sup> Известно, что такая точная функция полезности может быть представлена в виде одной-двух страниц кода на языке Lisp, Java или C++. Это означает, что она может быть представлена с помощью программы, которая находит точное решение для игры после каждого ее вызова. Но нас интересуют только аппроксиматоры функций, которые позволяют обойтись приемлемым объемом вычислений. В действительности может оказаться, что лучше найти с помощью обучения очень простой аппроксиматор функции и применять его в сочетании с некоторым объемом опережающего поиска. Но связанные с этим компромиссы в настоящее время еще недостаточно хорошо изучены.

этого подхода, укажем, что путем исследования только одного состояния из каждой группы по  $10^{44}$  возможных состояний в игре в нарды можно определить с помощью обучения функцию полезности, позволяющую программе играть не хуже любого игрока-человека [1499].

Оборотной стороной этого подхода, безусловно, является то, что с ним связана такая проблема: невозможность найти в выбранном пространстве гипотез какую-либо функцию, аппроксимирующую истинную функцию полезности достаточно хорошо. Как и во всем научном направлении индуктивного обучения, необходимо найти компромисс между размером пространства гипотез и потребностью во времени для определения с помощью обучения требуемой функции. С увеличением пространства гипотез растет вероятность того, что может быть найдена хорошая аппроксимация, но это означает, что сходимость также, скорее всего, будет достигаться более медленно.

Начнем с простейшего случая, в котором предусматривается непосредственная оценка полезности (см. раздел 21.2). При использовании функциональной аппроксимации определение такой оценки представляет собой пример **контролируемого обучения**. Например, предположим, что полезности для мира  $4 \times 3$  представлены с использованием простой линейной функции. Характеристиками квадратов являются их координаты  $x$  и  $y$ , поэтому получим следующее соотношение:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y \quad (21.9)$$

Таким образом, если  $(\theta_0, \theta_1, \theta_2) = (0.5, 0.2, 0.1)$ , то  $\hat{U}_\theta(1, 1) = 0.8$ . По результатам некоторой совокупности попыток будет получено множество выборочных значений  $\hat{U}_\theta(x, y)$ , после чего можно найти соответствие, наилучшее с точки зрения минимизации квадратичных ошибок, с помощью стандартной линейной регрессии (см. главу 20).

Применительно к обучению с подкреплением больше смысла имеет использование алгоритма оперативного обучения, который обновляет параметры после каждой попытки. Предположим, что осуществляется некоторая попытка, и суммарное вознаграждение, полученное начиная с квадрата  $(1, 1)$ , составляет 0.4. Такой результат наводит на мысль, что значение  $\hat{U}_\theta(1, 1)$ , составляющее в настоящее время 0.8, слишком велико и должно быть уменьшено. Как следует откорректировать параметры, чтобы добиться такого уменьшения? Как и в случае с обучением нейронной сети, запишем функцию ошибки и вычислим ее градиент по отношению к параметрам. Если  $u_j(s)$  — наблюдаемое суммарное вознаграждение, полученное от состояния  $s$  и дальше, вплоть до  $j$ -й попытки, то ошибка определяется как (половина) квадратов разностей прогнозируемой общей суммы и фактической общей суммы:  $E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2 / 2$ . Скорость изменения ошибки по отношению к каждому параметру  $\theta_i$  равна  $\partial E_j / \partial \theta_i$ , поэтому, чтобы изменить значение параметра в направлении уменьшения ошибки, необходимо вычислить следующее выражение:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha(u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \quad (21.10)$$

Это выражение называется **правилом Видроу-Хоффа**, или **дельта-правилом** для оперативного вычисления по методу наименьших квадратов. Применительно к линейному аппроксиматору функции  $\hat{U}_\theta(s)$  в уравнении 21.9 получаем три простых правила обновления:

$$\begin{aligned}\theta_0 &\leftarrow \theta_0 + \alpha(u_j(s) - \hat{U}_\theta(s)) \\ \theta_1 &\leftarrow \theta_1 + \alpha(u_j(s) - \hat{U}_\theta(s))x \\ \theta_2 &\leftarrow \theta_2 + \alpha(u_j(s) - \hat{U}_\theta(s))y\end{aligned}$$

Эти правила можно применить к примеру, в котором  $\hat{U}_\theta(1, 1)$  равно 0.8, а  $u_j(1, 1)$  равно 0.4. Все значения  $\theta_0$ ,  $\theta_1$  и  $\theta_2$  уменьшаются на коэффициент  $0.4\alpha$ , который уменьшает ошибку, относящуюся к квадрату (1, 1). Обратите внимание на то, что изменение значений  $\theta_i$  приводит также к изменению значений  $\hat{U}_\theta$  для всех прочих состояний! Именно это подразумевалось в приведенном выше утверждении, что функциональная аппроксимация позволяет агенту, проводящему обучение с подкреплением, обобщать свой опыт.

Мы рассчитываем на то, что агент будет проводить обучение быстрее, если он использует какой-то аппроксиматор функции, при условии, что пространство гипотез не слишком велико, но включает некоторые функции, характеризующиеся достаточно приемлемым соответствием истинной функции полезности. В упр. 21.7 предлагаются оценить производительность метода непосредственной оценки полезности, как с функциональной аппроксимацией, так и без нее. В мире  $4 \times 3$  действительно достигается заметное, однако не столь существенное увеличение производительности, прежде всего потому, что это пространство состояний очень мало. Достигнутое увеличение производительности становится намного более значительным в мире  $10 \times 10$  с вознаграждением +1 в квадрате (10, 10). Этот мир хорошо приспособлен для линейной функции полезности, поскольку истинная функция полезности является гладкой и почти линейной (см. упр. 21.10). А если вознаграждение +1 будет помещено в квадрат (5, 5), то истинная функция полезности будет больше напоминать по своей форме пирамиду, и попытка применения аппроксиматора функции, приведенного в уравнении 21.9, окончится крахом. Но не все потеряно! Напомним, что для линейной функциональной аппроксимации важно, чтобы функция линейно зависела от параметров. А сами характеристики могут представлять собой произвольные нелинейные функции от переменных состояния. Поэтому можно включить такой терм, как  $\theta_3 = \sqrt{(x - x_g)^2 + (y - y_g)^2}$ , измеряющий расстояние до цели.

Эти идеи можно применить столь же успешно к агентам, осуществляющим обучение по методу временной разности. Для этого достаточно откорректировать параметры, чтобы попытаться уменьшить временную разность между последовательными состояниями. Новые версии уравнений для метода TD и метода Q-обучения (21.3 и 21.8) приведены ниже. Уравнение для полезностей является следующим:

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \quad (21.11)$$

А для Q-значений используется следующее уравнение:

$$\theta_i \leftarrow \theta_i + \alpha [R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s)] \frac{\partial \hat{Q}_\theta(a, s)}{\partial \theta_i} \quad (21.12)$$

Можно показать, что эти правила обновления сходятся к ближайшей возможной<sup>5</sup> аппроксимации истинной функции, если аппроксиматор функции линейно зависит

<sup>5</sup> Это определение расстояния между функциями полезности является довольно формальным; см. [1515].

от параметров. К сожалению, при использовании нелинейных функций (таких как функции, задаваемые с помощью нейронных сетей) больше ничего нельзя гарантировать. Параметры могут увеличиваться до бесконечности и в некоторых очень простых случаях, даже несмотря на то, что в пространстве гипотез существуют приемлемые решения. Разработаны более сложные алгоритмы, позволяющие избежать этих проблем, но в настоящее время вся область обучения с подкреплением на основе общих аппроксиматоров функций продолжает оставаться тонким искусством.

Функциональная аппроксимация может также оказаться очень полезной при определении с помощью обучения модели среды. Напомним, что задача определения с помощью обучения модели для наблюдаемой среды представляет собой задачу контролируемого обучения, поскольку каждый следующий результат восприятия предоставляет информацию о результирующем состоянии. При этом может использоваться любой из методов контролируемого обучения, описанный в главе 18, с соответствующими поправками с учетом того факта, что необходимо определить с помощью прогноза полное описание состояния, а не просто булеву классификацию или единственное реальное значение. Например, если состояние определяется с помощью  $n$  булевых переменных, то необходимо найти с помощью обучения  $n$  булевых функций для прогнозирования всех переменных. А в случае частично наблюдаемой среды задача обучения становится гораздо более сложной. Если известно, какие скрытые переменные и какими причинными отношениями они связаны друг с другом и с наблюдаемыми переменными, то можно зафиксировать структуру динамической байесовской сети и воспользоваться алгоритмом EM для определения с помощью обучения ее параметров, как было описано в главе 20. А задачи выявления скрытых переменных и определения структуры модели с помощью обучения все еще остаются открытыми.

Теперь обратимся к примерам крупномасштабных приложений обучения с подкреплением. На основании этих примеров будет показано, что в случае использования функции полезности (следовательно, некоторой модели) модель обычно считается заданной. Например, при определении с помощью обучения функции оценки для нац общно предполагается, что допустимые ходы и их результаты известны заранее.

## Приложения методов обучения к ведению игр

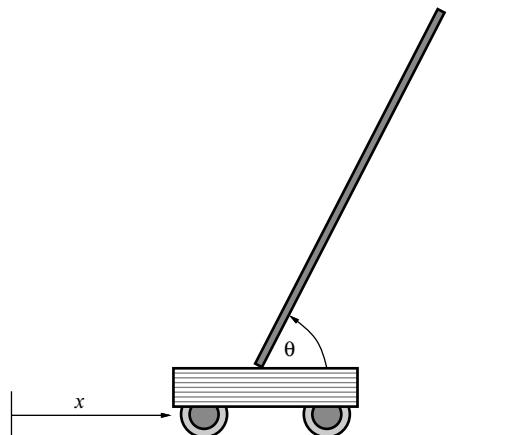
Первое реальное приложение метода обучения с подкреплением оказалось также первой практической применимой программой из всех возможных типов таких программ; речь идет о программе игры в шашки, написанной Артуром Самюэлом [1349], [1350]. Самюэл впервые использовал для оценки позиций взвешенную линейную функцию, в которой одновременно применялось до 16 термов. В его программе обновление весов осуществлялось на основе некоторой версии уравнения 21.11. Тем не менее методы, применяемые в его программе, имели определенные существенные отличия от современных методов. Первое отличие состояло в том, что для обновления весов использовалась разность между значениями для текущего состояния и зарезервированным значением, сформированным путем полного опера-жающего просмотра в дереве поиска. Такой подход оказался очень успешным, поскольку он равносителен подходу, в котором пространство состояний рассматривается с разными степенями детализации. Второе отличие состояло в том, что в программе Самюэла не использовались какие-либо наблюдаемые вознаграждения! Это означает, что значения терминальных состояний игнорировались. Таким образом, существует

вовала реальная возможность, что вычисления в программе не будут сходиться, или по крайней мере сходиться к стратегии, позволяющей выиграть, а не проиграть. Но автор программы сумел избежать подобного неблагоприятного развития событий, соблюдая такое требование, что вес материального преимущества должен всегда быть положительным. Замечательно то, что этого оказалось достаточно, чтобы всегда направлять программу в область пространства весов, соответствующую успешной игре в шашки.

Возможности методов обучения с подкреплением особенно наглядно показала система TD-Gammon Герри Тесауро [1499]. В одной из более ранних работ [1501] Тесауро попытался определить с помощью обучения некоторое представление функции  $Q(a, s)$  в виде нейронной сети непосредственно на примерах ходов, которым были присвоены относительные значения экспертом-человеком. Но, как оказалось, этот подход требовал исключительно высоких трудозатрат со стороны привлеченного эксперта. Применение такого подхода привело к созданию программы под названием Neurogammon, которая оказалась сильнее, чем другие программы игры в нарды, существовавшие в то время, но не выдерживала соревнования с опытными игроками в нарды. Проект TD-Gammon стал попыткой обучения программы исключительно в игре с самой собой. Единственный сигнал вознаграждения предоставился в конце каждой игры. Функция оценки была представлена в виде полносвязной нейронной сети с единственным скрытым слоем, содержащим 40 узлов. Программа TD-Gammon сумела в результате обучения достичь значительно более высокого уровня игры по сравнению с Neurogammon путем повторного применения уравнения 21.11, даже несмотря на то, что входное представление содержало только грубые оценки позиции на доске, без вычисленных характеристик. Для обучения программы потребовалось проведение около 200 000 учебных игр и две недели машинного времени. Хотя такое количество игр может показаться очень большим, оно фактически составляет лишь исчезающе малую долю пространства состояний. После того как к входному представлению были добавлены предварительно рассчитанные характеристики с применением сети, включающей 80 скрытых элементов, в результате 300 000 учебных игр удалось достичь уровня игры, сравнимого с уровнем трех ведущих людей-игроков во всем мире. Кит Вулси, ведущий игрок и аналитик, сказал: “У меня не было малейшего повода, чтобы усомниться в том, что эта программа оценивает позицию гораздо лучше по сравнению со мной”.

### Применение к управлению роботами

Постановка знаменитой задачи поиска равновесия системы **тележка-шест**, известной также под названием задачи **обратного маятника**, показана на рис. 21.6. Задача состоит в том, чтобы обеспечить управление положением тележки в направлении оси  $x$  таким образом, чтобы шест стоял примерно вертикально ( $\theta \approx \pi/2$ ), а сама тележка оставалась в пределах длины пути, как показано на этом рисунке. Такой с виду простой задаче посвящено свыше двух тысяч статей по обучению с подкреплением и теории управления. Задача с тележкой и шестом отличается от описанных выше задач тем, что переменные состояния  $x$ ,  $\theta$ ,  $\dot{x}$ , и  $\dot{\theta}$  являются непрерывными. С другой стороны, действия обычно остаются дискретными: таковыми являются рывки тележки влево или вправо в так называемом режиме **релейного управления**.



*Рис. 21.6. Постановка задачи уравновешивания длинного шеста, установленного на движущейся тележке. Контроллер, измеряющий значения  $x$ ,  $\theta$ ,  $x$  и  $\theta$ , рывками двигает тележку влево или вправо*

Одна из первых работ по обучению механической системы решению этой задачи была выполнена Мичи и Чамберсом [1046]. Разработанный ими алгоритм Boxes оказался способным поддерживать равновесие шеста больше одного часа после всего лишь 30 пробных попыток. Более того, в отличие от многих разработанных в дальнейшем систем, алгоритм Boxes был реализован с помощью реальной тележки и шеста, а не эмулятора. На первых порах в этом алгоритме применялась дискретизация четырехмерного пространства состояний с разбивкой на гиперкубы, называемые разработчиками *boxes*; отсюда произошло название алгоритма. После ввода алгоритма в действие выполнялись учебные попытки, продолжавшиеся до тех пор, пока не падал шест или тележка не достигала конца пути. Отрицательное подкрепление связывалось с заключительным действием в конечном гиперкубе, а затем распространялось в обратном направлении через всю последовательность действий. В дальнейшем было обнаружено, что применяемый способ дискретизации становится причиной возникновения некоторых проблем, когда вся эта экспериментальная установка инициализировалась в позиции, отличной от той, которая применялась при обучении (что свидетельствовало о недостаточно качественном обобщении). Лучшего обобщения и более быстрого обучения можно добиться, используя алгоритм, который адаптивно фрагментирует пространство состояний согласно наблюдаемому изменению в вознаграждении. В наши дни считается тривиальной задача уравновешивания трех обратных маятников, что значительно превосходит возможности большинства людей.

## 21.5. ПОИСК СТРАТЕГИИ

Последний подход к решению задач обучения с подкреплением, который рассматривается в этой главе, носит название **поиска стратегии**. В определенных отношениях метод, основанный на поиске стратегии, является самым простым из всех методов, рассматриваемых в данной главе; его идея состоит в том, что необходимо

продолжать корректировать стратегию до тех пор, пока связанная с ней производительность увеличивается, а после этого останавливаться.

Начнем с анализа самих стратегий. Напомним, что стратегией  $\pi$  называется функция, которая отображает состояния на действия. Нас прежде всего интересуют параметризованные представления  $\pi$ , которые имеют намного меньше параметров по сравнению с количеством состояний в пространстве состояний (как и в предыдущем разделе). Например, можно представить стратегию  $\pi$  в виде коллекции параметризованных Q-функций, по одной для каждого действия, и каждый раз предпринимать действие с наивысшим прогнозируемым значением:

$$\pi(s) = \max_a \hat{Q}_\theta(a, s) \quad (21.13)$$

Каждая Q-функция может представлять собой линейную функцию от параметров  $\theta$ , как показано в уравнении 21.9, или может быть и нелинейной функцией, например, представленной в виде нейронной сети. В таком случае поиск стратегии сводится к корректировке параметров  $\theta$  в целях улучшения стратегии. Следует отметить, что если стратегия представлена с помощью Q-функций, то поиск стратегии приводит к процессу определения Q-функций с помощью обучения. *Этот процесс не следует рассматривать как Q-обучение!* При Q-обучении с помощью функциональной аппроксимации алгоритм находит такое значение  $\theta$ , что функция  $\hat{Q}_\theta$  становится “близкой” к функции  $Q^*$  — оптимальной Q-функции. При поиске стратегии, с другой стороны, отыскивается значение  $\theta$ , которое приводит к достижению высокой производительности; найденные значения могут отличаться весьма существенно<sup>6</sup>. Еще одним примером, четко иллюстрирующим это различие, является случай, в котором  $\pi(s)$  вычисляется, допустим, с помощью опережающего поиска на глубину 10 с приближенной функцией полезности  $\hat{U}_\theta$ . Значение  $\theta$ , позволяющее обеспечить качественное ведение игры, может быть получено задолго до того, как удастся получить приближение  $\hat{U}_\theta$ , напоминающее истинную функцию полезности.

Одним из недостатков представления стратегий в том виде, как показано в уравнении 21.13, является то, что в случае дискретных действий стратегия не является непрерывной функцией своих параметров<sup>7</sup>. Это означает, что существуют такие значения  $\theta$ , при которых бесконечно малые изменения  $\theta$  вызывают резкое переключение стратегии с одного действия на другое. Вследствие этого могут также происходить не являющиеся непрерывными изменения значения стратегии, в результате чего поиск на основе градиента становится затруднительным. По этой причине в методах поиска стратегии часто используется **стochasticное представление стратегии**  $\pi_\theta(s, a)$ , которое задает вероятность выбора действия  $a$  в состоянии  $s$ . Одним из широко применяемых представлений такого типа является **функция softmax**:

$$\pi_\theta(s, a) = \exp(\hat{Q}_\theta(a, s)) / \sum_{a'} \exp(\hat{Q}_\theta(a', s))$$

<sup>6</sup> Стало тривиальным такое замечание, что приближенная Q-функция, определяемая соотношением  $\hat{Q}_\theta(a, s) = Q^*(a, s) / 10$ , обеспечивает оптимальную производительность, даже несмотря на то, что она весьма далека от  $Q^*$ .

<sup>7</sup> В случае непрерывного пространства действий стратегия может быть гладкой функцией от ее параметров.

Функция softmax становится почти детерминированной, если одно действие намного лучше по сравнению с другими, но она всегда позволяет получить дифференцируемую функцию от  $\theta$ , поэтому ценность стратегии (которая связана непрерывной зависимостью с вероятностями выбора действий) определяется дифференцируемой функцией  $\theta$ .

Теперь рассмотрим методы улучшения стратегии. Начнем с простейшего случая — детерминированной стратегии и детерминированной среды. В этом случае задача вычисления стратегии становится тривиальной — просто выполняется эта стратегия и регистрируются накопленные вознаграждения; полученные данные определяют **ценность стратегии**  $\rho(\theta)$ . В таком случае улучшение стратегии представляет собой стандартную задачу оптимизации, как описано в главе 4. В частности, можно проследить за вектором **градиента стратегии**  $\nabla_{\theta}\rho(\theta)$ , при условии, что значение  $\rho(\theta)$  является дифференцируемым. Другой вариант состоит в том, что можно проследовать за **эмпирическим градиентом** путем восхождения к вершине, т.е. вычисления изменений в стратегии в ответ на небольшие приращения в значениях каждого параметра. При соблюдении обычных предосторожностей такой процесс сходится к локальному оптимуму в пространстве стратегий.

Если же среда или стратегия является стохастической, задача становится более сложной. Предположим, что предпринимается попытка применить метод восхождения к вершине, для чего требуется сравнить  $\rho(\theta)$  и  $\rho(\theta + \Delta\theta)$  при некотором небольшом значении  $\Delta\theta$ . Проблема состоит в том, что суммарное вознаграждение при каждой попытке может существенно изменяться, поэтому оценки значения стратегии на основе небольшого количества попыток могут оказаться совершенно ненадежными, а еще более ненадежными становятся результаты, полученные при попытке сравнить две такие оценки. Одно из решений состоит в том, чтобы предпринять много попыток, измеряя дисперсию выборок и используя ее для определения того, достаточно ли много было сделано попыток, чтобы получить надежные данные о направлении улучшения для  $\rho(\theta)$ . К сожалению, такой подход является практически не применимым во многих реальных задачах, когда каждая попытка может оказаться дорогостоящей, требующей больших затрат времени, и, возможно, даже опасной.

В случае стохастической стратегии  $\pi_{\theta}(s, a)$  существует возможность получить несмещенную оценку для градиента  $\theta$ ,  $\nabla_{\theta}\rho(\theta)$ , соответствующего параметрам  $\theta$ , непосредственно по результатам попыток, выполненных при таких значениях параметра  $\theta$ . Для упрощения задачи выведем формулу такой оценки для простого случая непоследовательной среды, в которой вознаграждение предоставляется непосредственно после осуществления действия в начальном состоянии  $s_0$ . В таком случае значение стратегии является просто ожидаемым значением вознаграждения, поэтому имеет место следующее:

$$\nabla_{\theta}\rho(\theta) = \nabla_{\theta} \sum_a \pi_{\theta}(s_0, a) R(a) = \sum_a (\nabla_{\theta}\pi_{\theta}(s_0, a)) R(a)$$

Теперь можно применить простой пример, позволяющий аппроксимировать результаты этого суммирования с помощью выборок, сформированных на основании распределения вероятностей, определенного стратегией  $\pi_{\theta}(s_0, a)$ . Предположим,

что общее количество попыток равно  $N$ , а действием, предпринятым в  $j$ -й попытке, является  $a_j$ . В таком случае получим следующее:

$$\nabla_{\theta} \rho(\theta) = \sum_a \pi_{\theta}(s_0, a) \cdot \frac{(\nabla_{\theta} \pi_{\theta}(s_0, a)) R(a)}{\pi_{\theta}(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^N \frac{(\nabla_{\theta} \pi_{\theta}(s_0, a_j)) R(a_j)}{\pi_{\theta}(s_0, a_j)}$$

Поэтому истинный градиент значения стратегии аппроксимируется суммой термов, включающей градиент вероятности выбора действия при каждой попытке. Для последовательного случая это соотношение можно обобщить до такого соотношения для каждого посещенного состояния  $s$ :

$$\nabla_{\theta} \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N \frac{(\nabla_{\theta} \pi_{\theta}(s, a_j)) R_j(s)}{\pi_{\theta}(s, a_j)}$$

где  $a_j$  — действие, выполненное в состоянии  $s$  при  $j$ -й попытке;  $R_j(s)$  — суммарное вознаграждение, полученное, начиная от состояния  $s$  и дальше, при  $j$ -й попытке. Полученный в результате алгоритм называется Reinforce [1597]; обычно он является гораздо более эффективным по сравнению с восхождением к вершине, при котором используется большое количество попыток в расчете на каждое значение  $\theta$ . Тем не менее он все еще действует гораздо медленнее, чем абсолютно необходимо.

Рассмотрим следующее задание: даны две программы игры в очко<sup>8</sup>; необходимо определить, какая из них лучше. Один из способов выполнения этого задания состоит в организации игры каждой программы против стандартной программы, “сдающей карты”, в течение определенного количества раздач, с последующим сравнением выигрышей этих программ. Но этот подход связан с определенными проблемами, поскольку, как уже было сказано, выигрыш каждой программы изменяется в широких пределах в зависимости от того, получала ли она после каждой раздачи хорошие или плохие карты. Очевидным решением является заблаговременная выработка определенного количества раздач и множества карт, выдаваемых на руки. Благодаря этому устраняется ошибка измерения, связанная с различиями в полученных картах. Именно эта идея лежит в основе алгоритма Pegasus [1134]. Такой алгоритм является применимым в таких проблемных областях, для которых предусмотрен эмулятор, позволяющий повторно вырабатывать “случайные” результаты действий. Алгоритм функционирует по принципу заблаговременного формирования  $N$  последовательностей случайных чисел, каждая из которых может использоваться для прогона одного варианта опробования любой стратегии. Поиск стратегии осуществляется путем оценки каждой потенциальной стратегии с помощью одного и того же множества случайных последовательностей для определения результатов действия. Можно показать, что количество случайных последовательностей, требуемых для обеспечения качественной оценки значения любой стратегии, зависит только от сложности пространства стратегий, а не от сложности соответствующей проблемной области. Алгоритм Pegasus использовался для разработки эффективных стратегий в нескольких проблемных областях, включая автономный полет вертолета (рис. 21.7).

<sup>8</sup> Эта игра известна также под названием двадцать одно или блек джек.



*Рис. 21.7. Наложение изображений автономно управляемого вертолета, выполняющего очень сложный маневр “нос в круге”. Вертолет совершает полет под управлением стратегии, разработанной алгоритмом поиска стратегии Pegasus. Модель эмулятора была разработана путем наблюдений за результатами различных управляющих манипуляций реального вертолета; после этого алгоритм в течение одной ночи выполнял прогон на этой модели эмулятора. Был разработан целый ряд контроллеров для различных маневров. Во всех случаях достигнутая производительность была намного выше по сравнению с производительностью опытного пилота-человека, использующего дистанционное управление (фотография приведена с разрешения Эндрю Энджея (Andrew Ng))*

## 21.6. РЕЗЮМЕ

В данной главе рассматривалась задача обучения с подкреплением, решение которой позволяет агенту добиться успеха в неизвестной среде, пользуясь только полученными им результатами восприятия, а изредка также вознаграждениями. Обучение с подкреплением может рассматриваться как микроскопическая модель всей проблематики искусственного интеллекта, но для решения этой задачи применяется целый ряд упрощений, позволяющих быстрее добиться успеха. Основные идеи, изложенные в этой главе, перечислены ниже.

- Характер информации, которая должна быть получена в результате обучения, зависит от общего проекта агента. В данной главе рассматривались три основных проекта: проект, основанный на модели, в котором используется модель  $T$  и функция полезности  $U$ ; проект без модели, в котором применяется функция “действие–значение”, или Q-функция, и рефлексный проект, в котором используется стратегия  $\pi$ .
- Для определения полезностей с помощью обучения могут использоваться три перечисленных ниже подхода.
  1. При **непосредственной оценке полезности** применяется суммарное прогнозируемое наблюдаемое вознаграждение для заданного состояния в качестве

прямого свидетельства, позволяющего определить полезность данного состояния с помощью обучения.

2. В **адаптивном динамическом программировании** (Adaptive Dynamic Programming — ADP) с помощью обучения определяются модель и функция вознаграждения на основании наблюдений, а затем используется итерация по значениям или по стратегиям для получения полезностей или выявления оптимальной стратегии. В методе ADP обеспечивается оптимальное использование локальных ограничений, налагаемых на полезности состояний под влиянием структуры отношений соседства в рассматриваемой среде.
3. В методах **временной разности** (Temporal Difference — TD) обновляются оценки полезности для их согласования с состояниями-преемниками. Подход, основанный на использовании этих методов, может рассматриваться как простая аппроксимация подхода ADP, в котором не требуется модель для процесса обучения. Однако применение определенной в процессе обучения модели для выработки псевдорезультатов опытов способствует ускорению обучения.
  - Определение с помощью обучения функций “действие–значение”, или Q-функций, может быть организовано на основе подхода ADP или TD. При использовании метода TD в процессе Q-обучения не требуется модель ни на этапе обучения, ни на этапе выбора действия. Это позволяет упростить задачу обучения, но в принципе может привести к ограничению способности проводить обучение в сложных вариантах среды, поскольку агент не сможет моделировать результаты применения возможных стратегий.
  - Если от обучающегося агента требуется, чтобы он выбирал действия, пока еще продолжается обучение, то агенту приходится искать компромисс между оцениваемым значением этих действий и перспективами определения с помощью обучения новой полезной информации. Задача поиска точного решения такой проблемы исследования является неосуществимой, но некоторые простые эвристики позволяют добиться приемлемых результатов.
  - Если пространство состояний велико, то в алгоритмах обучения с подкреплением необходимо использовать приближенное функциональное представление для обобщения сведений о состояниях. Для обновления параметров таких представлений, как нейронные сети, можно непосредственно использовать информацию о временной разности.
  - Методы **поиска стратегии** применяются непосредственно к представлению стратегии в попытке улучшить ее с учетом наблюдаемой производительности. Изменчивость производительности в стохастической проблемной области представляет собой серьезную проблему; в случае моделируемых проблемных областей эту сложность можно преодолеть, заранее формируя случайные выборки.
  - Обучение с подкреплением позволяет избавиться от необходимости разрабатывать вручную стратегии управления, поэтому продолжает оставаться одной из наиболее активных областей исследований машинного обучения. Особенно ценными могут стать приложения этих подходов в робототехнике; для этого потребуются методы обеспечения действий в непрерывных, многомер-

ных, частично наблюдаемых вариантах среды, в которых успешное поведение может складываться из тысяч или даже миллионов примитивных действий.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Подход, основанный на обучении с подкреплением, предложил Тьюринг [1519], [1520], но он не был убежден в его общей эффективности и писал, что “использование наказаний и вознаграждений в лучшем случае может составлять лишь часть процесса обучения”. По-видимому, одним из первых успешных исследований по машинному обучению была работа Артура Самюэла [1349]. Хотя эта работа не была основана на теоретическом фундаменте и имела целый ряд недостатков, она содержала большинство современных идей в области обучения с подкреплением, включая применение временной разности и функциональной аппроксимации. Примерно в то же время исследователи в области теории адаптивного управления Видроу и Хофф [1587], опираясь на работу Хебба [638], занимались обучением простых сетей с помощью дельта-правила. (Долго существовавшие неправильные представления о том, что обучение с подкреплением является частью проблематики нейронных сетей, могло быть вызвано тем, что связь между этими научными областями установилась так рано.) Как метод обучения с подкреплением на основе аппроксиматора функции может также рассматриваться работа Мичи и Чамберса [1046], посвященная системе “тележка-шест”. Психологические исследования в области обучения с подкреплением начались гораздо раньше; хороший обзор по этой теме приведен в [653]. Непосредственные свидетельства того, как осуществляется обучение с подкреплением у животных, были получены в исследованиях поведения пчел, связанного с добычей пищи; достоверно обнаружен нейронный аналог структуры передачи сигнала вознаграждения в виде крупного нейронного образования, связывающего рецепторы органов взятия нектара непосредственно с двигательной корой [1070]. Исследования, в которых используется регистрация активности отдельной клетки, показали, что допаминовая система в мозгу приматов реализует нечто напоминающее обучение на основе стоимостной функции [1369].

Связь между обучением с подкреплением и марковскими процессами принятия решений была впервые отмечена в [1580], но разработка методов обучения с подкреплением в рамках искусственного интеллекта началась с исследований, проводимых в Массачусетском университете в начале 1980-х годов [76]. Хороший исторический обзор подготовлен Саттоном [1477]. Уравнение 21.3, приведенное в этой главе, представляет собой частный случай общего алгоритма TD( $\lambda$ ) Саттона при  $\lambda=0$ . В алгоритме TD( $\lambda$ ) обновляются значения всех состояний в последовательности, ведущей вплоть до каждого перехода, на величину, которая уменьшается в зависимости от  $\lambda^t$  для состояний, отстоящих на  $t$  шагов в прошлое. Алгоритм TD(1) идентичен правилу Видроу-Хоффа, или дельта-правилу. Бойян [162], опираясь на [170], доказал, что в алгоритме TD( $\lambda$ ) и связанных с ним алгоритмах результаты, полученные опытным путем, используются неэффективно; по сути они представляют собой алгоритмы оперативной регрессии, которые сходятся гораздо медленнее, чем алгоритмы автономной регрессии. Предложенный Бойяном алгоритм LSTD( $\lambda$ ) представляет собой оперативный алгоритм, позволяющий достичь таких же результатов, как и алгоритм автономной регрессии.

Применение сочетания метода обучения на основе временной разности с методом формирования моделируемых результатов опытов с помощью модели было предложено в архитектуре Саттона Dyna [1479]. Идея выметания с учетом приоритетов была предложена независимо Муром и Аткесоном [1077], а также Пенгом и Уильямсом [1203]. Метод Q-обучения был разработан в докторской диссертации Уоткинса [1558].

Задачи с n-рукими бандитами, которые моделируют задачу исследования непоследовательных решений, были глубоко проанализированы в [116]. Оптимальные стратегии исследования для нескольких постановок задач могут быть получены с помощью метода, называемого **индексами Гиттинса** [561]. Целый ряд методов исследования, применимых для решения задач последовательного принятия решений, обсуждается в [74]. В [171] и [785] описаны алгоритмы, позволяющие исследовать неизвестные варианты среды и гарантированные сходимость к стратегиям, близким к оптимальным, за полиномиальное время.

Истоки идей по применению функциональной аппроксимации в обучении с подкреплением можно найти в работах Самюэла, который использовал линейные и нелинейные функции оценки, а также методы выбора характеристик для уменьшения пространства характеристик. В дальнейшем были разработаны такие методы, как ~~CMAC~~ CMAC (Cerebellar Model Articulation Controller) [12], который по сути сводится к использованию суммы перекрывающихся локальных ядерных функций, и ассоциативные нейронные сети [75]. В настоящее время в качестве аппроксиматоров функций наиболее широко используются нейронные сети. Наиболее известным приложением является программа TD-Gammon [1499], [1500], которая описывалась в данной главе. Одной из существенных проблем, возникающих при использовании обучающихся по методу TD агентов, основанных на нейронной сети, является то, что они, как правило, забывают полученные раньше результаты опытов, особенно касающиеся тех частей пространства состояний, которых они стали избегать после приобретения достаточной компетентности. Этот недостаток может приводить к катастрофическому отказу, если снова возникают подобные обстоятельства. Такую проблему позволяет устранить функциональная аппроксимация с помощью **обучения на основе экземпляра** (instance-based learning) [477], [1159].

Тема, касающаяся сходимости алгоритмов обучения с подкреплением, в которых применяется функциональная аппроксимация, требует чрезвычайно формального освещения. В случае использования аппроксиматоров линейных функций результаты обучения по методу TD неизменно улучшались [338], [1477], [1515], но было показано, что при использовании нелинейных функций обнаруживаются некоторые примеры отсутствия сходимости (для ознакомления с этой темой см. [1515]). В [1172] описан новый тип алгоритма обучения с подкреплением, который сходится при использовании любой формы аппроксиматора функций, при условии, что для наблюдаемых данных может быть найдена аппроксимация с наилучшим соответствием.

Методы поиска стратегии вышли на передний план под влиянием исследований Уильямса [1597], который разработал семейство алгоритмов Reinforce. Дальнейшие исследования, описанные в [86], [981] и [1478], позволили усилить и обобщить результаты достижения сходимости в методе поиска стратегии. Алгоритм Pegasus был предложен Энджи и Джорданом [1134], но аналогичные методы изложены в докторской диссертации Ван Роя [1535]. Как упоминалось в этой главе, производительность стохастической стратегии представляет собой непрерывную функцию от ее

параметров, что способствует применению методов поиска с учетом градиента. Это — не единственное преимущество указанных методов; в [721] доказано, что стохастические стратегии обычно функционируют в частично наблюдаемых вариантах среды лучше, чем детерминированные стратегии, если те и другие ограничиваются действиями, основанными на текущих результатах восприятия. (Одна из причин этого состоит в том, что стохастическая стратегия с меньшей вероятностью “заходит в тупик”, встретив какое-то невидимое препятствие.) Кроме того, в главе 17 было указано, что оптимальные стратегии в частично наблюдаемых задачах MDP представляют собой детерминированные функции от доверительного состояния, а не от текущих результатов восприятия, поэтому можно рассчитывать на получение еще лучших результатов с помощью сложения за доверительным состоянием с использованием методов **фильтрации**, приведенных в главе 15. К сожалению, пространство доверительных состояний является многомерным и непрерывным, и еще не разработаны эффективные алгоритмы обучения с подкреплением на основе доверительных состояний.

Реальные варианты среды также характеризуются невероятной сложностью с точки зрения количества примитивных действий, требуемых для достижения значимых вознаграждений. Например, работу, играющему в футбол, могут потребоваться сотни тысяч отдельных движений ног для того, чтобы забить мяч. Одним из широко применяемых методов, который первоначально использовался при обучении животных, является так называемый метод **формирования вознаграждения** (reward shaping). Он предусматривает предоставление агенту дополнительных вознаграждений за то, что он “добивается успехов”. Что касается футбола, то такие вознаграждения могут предоставляться за удар по мячу или за отправку мяча в сторону ворот. Подобные вознаграждения позволяют существенно повысить скорость обучения, а сам способ их предоставления является очень простым, но существует опасность того, что агент обучится максимизации таких псевдовознаграждений и не будет стремиться к истинным вознаграждениям; например, многочисленных контактов с мячом можно добиться, стоя рядом с мячом и “совершая колебательные движения”. В [1133] показано, что агент все равно будет искать с помощью обучения оптимальную стратегию, при условии, что псевдовознаграждение  $F(s, a, s')$  удовлетворяет соотношению  $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ , где  $\Phi$  — произвольная функция от состояния. Функцию  $\Phi$  можно составить таким образом, чтобы она отражала все желательные аспекты состояния, такие как достижение подцелей или уменьшение расстояния до целевого состояния.

Выработку сложных вариантов поведения можно также обеспечить с помощью методов **иерархического обучения с подкреплением**, которые представляют собой попытку решать задачи на нескольких уровнях абстракции, во многом аналогично методам **планирования HTN**, описанным в главе 12. Например, цель “забить мяч” можно разделить на подцели “овладеть мячом”, “приближаясь к воротам, обвести противников” и “ударить по воротам”, а каждая из этих подцелей может быть дополнительно разделена на двигательные варианты поведения еще более низкого уровня. Фундаментальный результат в этой области получен Форестьером и Варайя [481], которые доказали, что варианты поведения низкого уровня с произвольной сложностью можно рассматривать как примитивные действия (хотя и учитывая при этом то, что они могут потребовать разного количества времени) с точки зрения вариантов поведения более высокого уровня, в которых они вызываются. В современ-

ных подходах [32], [397], [1177], [1478] эти результаты используются для создания методов, позволяющих предоставить агенту **частичную программу**, которая ограничивает поведение агента так, чтобы оно имело конкретную иерархическую структуру. После этого применяется обучение с подкреплением, чтобы агент определил наилучший вариант поведения, совместимый с этой частичной программой. Сочетание методов функциональной аппроксимации, формирования вознаграждения и иерархического обучения с подкреплением может стать основой успешного подхода к решению крупномасштабных задач.

Хорошим исходным пунктом для дальнейшего изучения литературы по этой теме может служить обзор [759]. В книге Саттона и Барто [1480], двух основоположников этой области, изложение сосредоточено на архитектурах и алгоритмах, а также показано, как методы обучения с подкреплением подпитываются идеями в области обучения, планирования и осуществления действий. В немного более формальной работе [121] приведено строгое изложение основ теории динамического программирования и стохастической сходимости. Статьи по обучению с подкреплением часто публикуются в журналах *Machine Learning* и *Journal of Machine Learning Research*, а также в материалах конференций *International Conferences on Machine Learning* и семинаров *Neural Information Processing Systems*.

## УПРАЖНЕНИЯ

- 21.1.** Реализуйте пассивного обучающегося агента, действующего в простой среде, такой как мир  $4 \times 3$ . Для случая первоначально неизвестной модели среды сравните производительность обучения с помощью алгоритмов непосредственной оценки полезности, TD и ADP. Проведите сравнение оптимальной стратегии и некоторых случайно выбранных стратегий. Применительно к какой из них быстрее всего сходятся оценки полезности? Что происходит при увеличении размеров среды? (Опробуйте варианты среды с препятствиями и без препятствий.)
- 21.2.** В главе 17 была определена **правильная стратегия** для некоторой задачи MDP как стратегия, позволяющая достичь терминального состояния. Покажите, что для пассивного агента ADP возможна такая ситуация, что он найдет с помощью обучения модель перехода, для которой его стратегия  $\pi$  является неправильной, даже если  $\pi$  правильна для истинной задачи MDP; при использовании таких моделей этап определения значения может окончиться неудачей, если  $\gamma=1$ . Покажите, что такая проблема не может возникнуть, если этап определения значения применяется к модели, создаваемой с помощью обучения, только в конце каждой попытки.
- 21.3.** Начиная с проекта пассивного агента ADP, модифицируйте его так, чтобы в нем использовался приближенный алгоритм ADP, как описано в настоящей главе. Выполните это задание за два описанных ниже этапа.
  - a)** Реализуйте приоритетную очередь для внесения корректировок в оценки полезностей. После корректировки данных для некоторого состояния все его предшественники также становятся кандидатами на проведение корректировки и должны быть внесены в очередь. Очередь инициализирует-

ся с учетом состояния, из которого произошел самый последний переход. Предусмотрите возможность использования только фиксированного количества корректировок.

- 6) Проведите эксперименты с использованием различных эвристик для упорядочения приоритетной очереди, исследуя их влияние на скорость обучения и продолжительность вычислений.
- 21.4.** В методе непосредственной оценки полезности, описанном в разделе 21.2, используются различные терминальные состояния для обозначения конца каждой попытки. Как можно модифицировать этот метод для применения в вариантах среды с обесцениваемыми вознаграждениями и без терминальных состояний?
- 21.5.** Как можно использовать алгоритм определения значения для вычисления ожидаемых потерь, испытываемых агентом, который применяет заданное множество оценок полезностей  $U$  и оцениваемую модель  $M$ , по сравнению с агентом, использующим правильные значения?
- 21.6.** Приспособьте мир пылесоса (см. главу 2) для обучения с подкреплением, включив в него вознаграждения за то, что пылесос собирает каждый фрагмент мусора, отправляется в свой исходный квадрат и отключается. Сделайте этот мир доступным, предоставив агенту соответствующие результаты восприятия. После этого проведите эксперименты с различными агентами, действующими на основе обучения с подкреплением. Является ли функциональная аппроксимация обязательным условием успеха? Какого рода аппроксиматор может применяться в этом приложении?
- 21.7.** Реализуйте проект агента, исследующего свою среду и действующего на основе обучения с подкреплением, в котором используются непосредственные оценки полезностей. Подготовьте две версии — с табличным представлением и с применением аппроксиматора функции, показанного в уравнении 21.9. Сравните их производительность в трех вариантах среды, описанных ниже.
- a) Мир  $4 \times 3$ , описанный в данной главе.
  - б) Мир  $10 \times 10$  без препятствий и с вознаграждением +1 в квадрате  $(10, 10)$ .
  - в) Мир  $10 \times 10$  без препятствий и с вознаграждением +1 в квадрате  $(5, 5)$ .
- 21.8.** Запишите уравнения обновления параметров для метода обучения TD со следующим условием:
- $$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}$$
- 21.9.** Составьте список применимых характеристик для стохастических миров с решетками (обобщений мира  $4 \times 3$ ), которые содержат множество препятствий и множество терминальных состояний с вознаграждением -1 или +1.
- 21.10.** Вычислите истинную функцию полезности и наилучшую линейную аппроксимацию в точках  $x$  и  $y$  (как описано в уравнении 21.9) для перечисленных ниже вариантов среды.
- a) Мир  $10 \times 10$  с единственным терминальным состоянием +1 в квадрате  $(10, 10)$ .

- 6) Как и в упр. 21.10, а, но с дополнительным терминальным состоянием – 1 в квадрате (10, 1).
- в) Как и в упр. 21.10, б, но с дополнительными препятствиями в 10 квадратах, выбранных случайным образом.
- г) Как и в упр. 21.10, б, но с размещением стены, простирающейся от квадрата (5, 2) до квадрата (5, 9).
- д) Как и в упр. 21.10, а, но с терминальным состоянием в квадрате (5, 5).

Действия представляют собой детерминированные движения в четырех направлениях. В каждом случае сравните результаты с использованием трехмерных графиков. Для каждой среды предложите дополнительные характеристики (кроме  $x$  и  $y$ ), которые позволили бы улучшить эту аппроксимацию, и продемонстрируйте полученные результаты.

- 21.11.** Дополните стандартную среду ведения игры (см. главу 6) для включения в нее сигнала вознаграждения. Поместите в эту среду два агента, действующих на основе обучения с подкреплением (они, безусловно, могут иметь общую программу агента), и вынудите их играть друг против друга. Примените обобщенное правило обновления TD (уравнение 21.11) для обновления функции оценки. Вам может потребоваться вначале применить простую функцию оценки с линейными весами и простую игру, такую как крестики-нолики.
- 21.12.** Реализуйте алгоритмы Reinforce и Pegasus и примените их к миру  $4 \times 3$ , используя выбранное вами семейство стратегий. Прокомментируйте полученные результаты.
- 21.13.** Исследуйте проблему применения идей обучения с подкреплением для моделирования поведения людей и животных.
- 21.14.** Может ли обучение с подкреплением служить подходящей абстрактной моделью для эволюции? Какая связь существует (и существует ли она вообще) между жестко закрепленными сигналами вознаграждения и эволюционной пригодностью?

## Часть VII

# ОБЩЕНИЕ, ВОСПРИЯТИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ

Общение	1046
Вероятностная обработка лингвистической информации	1100
Восприятие	1139
Вероятностная обработка лингвистической информации	1179

## 22 ОБЩЕНИЕ

*В данной главе показано, почему агентам может потребоваться обмен сообщениями, несущими информацию, и как они могут это сделать.*

В роще среди саванны в Национальном парке Амбосели у подножья горы Килиманджаро царит полумрак. Группа мартышек-верветок сосредоточенно добывает пищу, и вдруг одна из них издает громкий лающий звук. Остальные члены группы распознают этот звук как предупреждение о появлении леопарда (в отличие от короткого кашля, используемого для предупреждения о появлении орлов, или стрекотания, обозначающего присутствие змей) и быстро взбираются на деревья. Верветка успешно выполнила акт общения с группой.

✧ **Общение** — это целенаправленный обмен информацией, осуществляемый путем предъявления и восприятия ✧ **знаков**, которые выбраны из совместно используемой системы знаков, принятых в соответствии с некоторым соглашением. Большинство животных используют знаки для представления важных сообщений: указание о наличии пищи, предупреждение о появлении хищника, требование приблизиться, требование удалиться, призыв к спариванию. В частично наблюдаемом мире общение может помочь агентам действовать более успешно, поскольку они получают возможность усваивать информацию, полученную другими с помощью наблюдения или логического вывода.

Отличием людей от животных является то, что они пользуются сложной системой обмена структурированными сообщениями, известной как ✧ **язык**, которая позволяет людям передавать друг другу большую часть того, что им стало известно о мире. Хотя шимпанзе, дельфины и другие млекопитающие показали наличие у них словарей, состоящих из сотен знаков, и обнаружили определенные способности связывать их в цепочки, только люди способны надежно общаться с помощью неограниченного количества качественно различных сообщений.

Безусловно, существуют и другие атрибуты, которые можно рассматривать как уникальные особенности людей: больше ни один вид живых существ не носит одежду, не создает произведения изобразительного искусства и не смотрит телевизор по три часа в день. Но когда Тьюринг предложил свой тест (см. раздел 1.1), он прежде всего исходил из способности владеть языком, поскольку язык тесно связан с мышлением. В настоящей главе приведено объяснение того, как действует общающийся агент, и дано описание небольшого подмножества английского языка.

## 22.1. ОБЩЕНИЕ КАК ДЕЙСТВИЕ

Одним из действий, доступных для любого агента, является выработка языковых сообщений. Такой процесс называется **речевым актом**. В этом термине понятие “речь” используется в таком же смысле, как и в выражении “речь идет о том-то”, а не означает просто “произнесение слов”, поэтому речевыми актами считаются любые действия, позволяющие составить и передать некоторое сообщение, например, с использованием электронной почты, сигнальных флагков или жестов. В естественном языке нет нейтрального слова для обозначения агента, вырабатывающего языковое сообщение с помощью средств, имеющихся в его распоряжении, поэтому в данной главе для описания любых возможных способов общения будут использоваться в качестве универсальных такие термины, как **отправитель** речевого сообщения, **получатель** речевого сообщения и **фрагмент речи**. Кроме того, для обозначения знака любого рода, который принято использовать в общении, будет служить термин **слово**.

Но по каким причинам агенту приходится заботиться о том, чтобы произвести речевой акт, когда он мог бы просто выполнить “обычное” действие? Как было показано в главе 12, агенты в мультиагентной среде могут использовать общение для выработки совместных планов. Например, группа агентов, исследующих мир вампуза вместе, приобретает значительные преимущества (общие и индивидуальные), получая возможность выполнять описанные ниже действия.

- Запрашивать других агентов о конкретных особенностях мира, в котором они существуют. Это действие обычно осуществляется путем постановки вопросов: “Ты где-то слышал запах вампуса?”
- Информировать друг друга об этом мире. Такая задача осуществляется путем составления описательных предложений: “Здесь, в квадрате [3,4], чувствуется ветерок”. Еще одним способом информирования является ответ на вопрос.
- Требовать от других агентов выполнения действий: “Пожалуйста, помоги мне отнести золото”. Иногда считается более вежливым **непрямой речевой акт** (требование в форме утверждения или вопроса): “Я охотно воспользовался бы чьей-то помощью, когда нужно будет отнести этот груз”. Агент, обладающий властью, может давать команды (“Альфа — направо, Браво и Чарли — налево”), а агент, обладающий силой, может высказать угрозу (“Отдай мне золото, иначе...”). Речевые акты такого рода, вместе взятые, называются **директивами**.
- Подтверждать согласие на выполнение требования: “Хорошо”.
- Предлагать варианты или выражать готовность участвовать в плане: “Я застрелил вампуса; вы заберете золото”.

Все речевые акты воздействуют на мир, заставляя колебаться молекулы воздуха (или оказывая эквивалентное воздействие на какой-то другой носитель информации); благодаря этому они изменяют мыслительное состояние и в конечном итоге будущие действия других агентов. Следствием речевых актов некоторых типов становится передача информации получателю в расчете на то, что эта информация повлияет должным образом на принятие решений получателем. Другие типы речевых актов нацелены более непосредственно на выполнение получателем каких-то действий. Еще один класс речевых актов, **декларативные** речевые акты, по-видимому,

оказывает более прямое воздействие на мир, как в случае произнесения слов: “Объявляю вас мужем и женой” или “Выпала тройка, и ваша игра окончена”. Безусловно, этот эффект достигается путем создания или подтверждения сложной сети мыслительных состояний среди участвующих агентов: вступление в брак и выход из игры — это состояния, обусловленные главным образом соблюдением некоторого соглашения, а не “физическими” свойствами мира.

Задача общающегося агента состоит в принятии решения в отношении того, когда потребуется речевой акт того или иного рода и какой речевой акт из всех возможных актов будет правильным. Проблема понимания воспринятых речевых актов в большей степени подобна другим проблемам **понимания**, таким как понимание смысла изображений или диагностирование заболеваний. Нам предъявляется множество неоднозначных входных сигналов, а исходя из него мы должны восстановить картину событий, чтобы определить, какое состояние мира могло привести к созданию этих входных данных. Но поскольку речь — это запланированное действие, то ее понимание связано также с распознаванием плана.

## Основные понятия языка

**Формальный язык** определяется как (возможно бесконечное) множество **строк**. Каждая строка представляет собой конкатенацию **терминальных символов**, иногда называемых *словами*. Например, в языке логики первого порядка терминальные символы включают  $\wedge$  и  $P$ , типичной строкой является “ $P \wedge Q$ ”, а строка “ $P Q \wedge$ ” не считается элементом этого языка. Формальные языки, такие как логика первого порядка и Java, имеют строгие математические определения. В этом они отличаются от **естественных языков**, таких как китайский, датский и английский, которые не имеют строгого определения, но совместно используются сообществом говорящих на них людей. Но в этой главе будет предпринята попытка трактовать естественные языки так, как если бы они были формальными языками, хотя авторы признают, что эта попытка провести между ними аналогию не будет идеальной.

**Грамматика** — это конечное множество правил, которое определяет язык. Формальные языки всегда имеют официально утвержденную грамматику, описанную в руководствах или учебниках. Естественные языки не имеют официально утвержденной грамматики, но лингвисты стремятся раскрыть свойства языка в процессе научного исследования, а затем узаконить свои открытия в грамматике. До сих пор еще ни одному лингвисту не удалось добиться в этом полного успеха. Следует отметить, что лингвисты — это ученые, пытающиеся дать определение естественному языку в том виде, в каком он есть. Но некоторые специалисты берут также на себя роль распространителей норм грамматики и пытаются диктовать, каким должен быть язык. Они создают правила, подобные тому, что “нельзя применять инфинитив с отделенной частицей” (*split infinitive*), а эти правила иногда публикуются в руководствах по языковому стилю, но оказывают очень малое влияние на то, как фактически используется язык.

И в формальных, и в естественных языках с каждой допустимой строкой связан смысл, или **семантика**. Например, в языке арифметики может быть предусмотрено правило, указывающее, что если “ $X$ ” и “ $Y$ ” — выражения, то “ $X+Y$ ” — также выражение, а его семантикой является сумма  $X$  и  $Y$ . В естественных языках важно также понимать  **pragmaticу** строки — фактический смысл строки, как речи, высказан-

ной в данной конкретной ситуации. Смысл заложен не только в самих словах, но и в интерпретации этих слов в сложившихся обстоятельствах.

Большинство формальных систем представления грамматических правил основано на идее **структуре словосочетаний**, согласно которой строки состоят из подстрок, называемых **словосочетаниями**, которые относятся к различным категориям. Например, словосочетания “the wumpus”, “the king” и “the agent in the corner” представляют собой примеры категории **именного словосочетания**, или сокращенно *NP* (Noun Phrase). Есть две причины, по которым целесообразно классифицировать словосочетания именно таким образом. Во-первых, словосочетания обычно соответствуют естественным семантическим элементам, из которых можно конструировать смысл фрагмента; например, именные словосочетания указывают на объекты в рассматриваемом мире. Во-вторых, классификация словосочетаний позволяет описывать строки, допустимые в данном языке. В частности, можно утверждать, что любое из именных словосочетаний может комбинироваться с **глагольным словосочетанием** (или сокращенно *VP* — Verb Phrase), таким как “is dead” (мертв), для формирования словосочетания, относящегося к категории **предложения** (или сокращенно *S* — Sentence). Без таких вспомогательных понятий, как именное словосочетание и глагольное словосочетание, было бы трудно объяснить, почему строка, состоящая из слов “the wumpus is dead”, представляет собой предложение, а “wumpus the dead is” — нет.

Такие имена категорий, как *NP*, *VP* и *S*, называются **нетерминальными символами**. В формальных грамматиках нетерминальные символы определяются с использованием **правил подстановки**. Авторы приняли в качестве системы обозначений для правил подстановки форму Бэкуса-Наура (Backus-Naur Form — BNF), которая описана в приложении Б на с. 1297. В этой системе обозначений смысл приведенного ниже правила состоит в том, что конструкция *S* может состоять из любой конструкции *NP*, за которой следует любая конструкция *VP*:

$$S \rightarrow NP\ VP$$

### ПОРОЖДАЮЩАЯ СПОСОБНОСТЬ

Грамматические формальные системы можно классифицировать по их **порождающей способности** — по тому множеству языков, которое они позволяют представить. Ноам Хомский [251] описал четыре класса грамматических формальных систем, которые отличаются только по форме правил подстановки. Эти классы можно расположить в виде иерархии, в которой каждый класс может использоваться для описания всех языков, которые могут быть описаны с помощью менее мощного класса, а также некоторых дополнительных языков. Ниже приведен список классов этой иерархии, в котором вначале приведен наиболее мощный класс.

- **Рекурсивно перечислимые** грамматики имеют неограниченные правила; по обе стороны правил подстановки может находиться любое количество терминальных и нетерминальных символов, как в правиле  $A\ B \rightarrow C$ . Эти грамматики по своей выразительной мощи эквивалентны машинам Тьюринга.
- **Контекстно-зависимые грамматики** ограничены лишь тем, что в правой части правил должно находиться по меньшей мере столько же символов, сколько и в левой части. Определение “контекстно-зависимый” связано с

тем фактом, что в правиле, подобном  $A \ S \ B \rightarrow A \ X \ B$ , указано, что вместо символа  $S$  может быть выполнена подстановка символа  $X$  в контексте предшествующего символа  $A$  и следующего символа  $B$ . Контекстно-зависимые грамматики способны представлять такие языки, как  $a^n b^n c^n$  (последовательность из  $n$  копий  $a$ , за которой следует такое же количество копий  $b$ , а затем  $c$ ).

- В **контекстно-свободных грамматиках** (или сокращенно **CFG** — Context-Free Grammar) левая часть каждого правила состоит из одного нетерминального символа. Таким образом, каждое правило обеспечивает подстановку вместо нетерминального символа правой части правила в любом контексте. Грамматики CFG широко применяются для описания естественных языков и представления в виде программ грамматик формальных языков, хотя теперь широко признано, что некоторые естественные языки включают конструкции, которые не являются контекстно-свободными [1242]. Контекстно-свободные грамматики позволяют представить язык  $a^n b^n$ , но не  $a^n b^n c^n$ .
- **Регулярные** грамматики представляют собой наиболее ограниченный класс. Каждое правило имеет один нетерминальный символ в левой части и терминальный символ, за которым может следовать или не следовать нетерминальный символ, в правой части. Регулярные грамматики эквивалентны по своей выразительной мощи конечным автоматам. Они плохо приспособлены для определения языков программирования, поскольку не позволяют представить такие конструкции, как сбалансированные открывающие и закрывающие скобки (один из вариантов языка  $a^n b^n$ ). Самое большее, что они позволяют представить, — это язык  $a^* b^*$ , последовательность, состоящую из любого количества символов  $a$ , за которым следует любое количество символов  $b$ .

Грамматики, расположенные выше в этой иерархии, имеют большую выразительную мощь, но алгоритмы для работы с ними являются менее эффективными. Вплоть до середины 1980-х годов усилия лингвистов были в основном сосредоточены на контекстно-свободных и контекстно-зависимых языках. Но в дальнейшем стали шире применяться регулярные грамматики, что было вызвано необходимостью очень быстро обрабатывать мегабайты и гигабайты текста в оперативном режиме, даже за счет менее полного анализа. Как сказал Фернандо Перейра: “Чем старше я становлюсь, тем ниже спускаюсь по иерархии Хомского”. Чтобы узнать, что он имел в виду, сравните его книгу, которая вышла в 1980 году [1208], с книгой, выпущенной в 2002 году [1069].

## Составные этапы общения

Типичный эпизод общения, в котором отправитель сообщения  $S$  желает проинформировать получателя сообщения  $H$  об истинности высказывания  $P$  с использованием слов  $W$ , состоит из семи описанных ниже процессов.

- **Намерение.** В какой-то момент времени отправитель  $S$  решает, что есть некоторое высказывание  $P$ , которое следует сообщить получателю  $H$ . В качестве

примера предположим, что отправитель хочет дать знать получателю о том, что вампуса больше нет в живых.

- **Выработка.** Отправитель составляет план преобразования высказывания  $P$  во фрагмент речи, обеспечивающий высокую вероятность того, что получатель после восприятия этого фрагмента речи в текущей ситуации сможет восстановить логическим путем смысл высказывания  $P$  (или чего-то близкого к нему). Предположим, что отправитель способен составить предложение из слов “The wumpus is dead”, и обозначим эти слова как  $W$ .
- **Синтез.** Отправитель производит физическую реализацию  $W'$  слов  $W$ . Это может выражаться в нанесении чернил на бумагу, создании колебаний воздуха или воздействии на какой-то другой носитель информации. На рис. 22.1 показано, что агент синтезирует строку звуков  $W'$ , записанную в фонетическом алфавите, который определен на с. 759: “[thaxwahmpaxsihzdehd]”. Слова сливаются друг с другом; это типично для быстро произносимой речи.

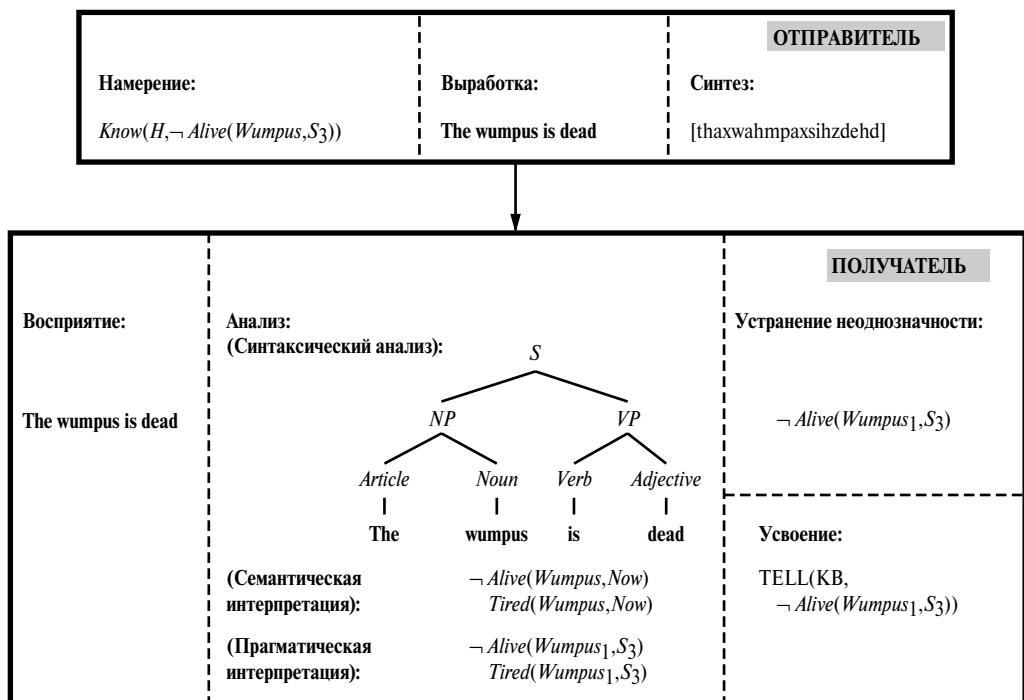


Рис. 22.1. Семь процессов, связанных с общением, которые показаны на примере предложения “The wumpus is dead” (Вампус мертв)

- **Восприятие.** Получатель  $H$  воспринимает физическую реализацию  $W'$  как  $W_2'$  и расшифровывает ее в виде слов  $W_2$ . Если носитель информации предназначен для передачи звуковой речи, такой этап восприятия называется **распознаванием речи**, а если носитель информации обеспечивает создание двухмерных отображений знаков, такой этап называется **оптическим распознаванием символов**. Оба эти способа распознавания в 1990-х годах перешли из

области действий, непостижимых по своей сложности, в сферу повседневного использования, в основном благодаря повышению вычислительной мощи настольных компьютеров.

- **↗ Анализ.** Получатель  $H$  приходит к логическому выводу, что слова  $W_2$  имеют возможные толкования  $P_1, \dots, P_n$ . Авторы подразделяют анализ на три основные части: синтаксическая интерпретация (или синтаксический анализ), семантическая интерпретация и прагматическая интерпретация. **↗ Синтаксический анализ** — это процесс построения **дерева синтаксического анализа** для входной строки (см. рис. 22.1). Внутренние узлы дерева синтаксического анализа представляют словосочетания, а листовые узлы — слова. **↗ Семантическая интерпретация** — это процесс извлечения смысла фрагмента речи как выражения в некотором языке представления. На рис. 22.1 показаны две возможные семантические интерпретации: то, что вампира нет в живых, и то, что он полностью обессилел (в английском языке слово “dead” может иметь в разговорной речи и этот смысл). Фрагменты речи с несколькими возможными интерпретациями называются **неоднозначными**. В **↗ прагматической интерпретации** учитывается тот факт, что одни и те же слова могут иметь разный смысл в разных ситуациях. Синтаксическая интерпретация может рассматриваться как функция от одного параметра (строки), а прагматическая интерпретация — как функция от фрагмента речи и контекста, или ситуации, в которой эта строка была сформирована. В этом примере прагматическая интерпретация сводится к выполнению двух действий: замене константы  $Now$  константой  $S_3$ , которая обозначает текущую ситуацию, и замене константы  $Wumpus$  константой  $Wumpus_1$ , которая обозначает того единственного вампира, который, как известно, находится в этой пещере. Вообще говоря, прагматическая интерпретация способна внести гораздо больший вклад в окончательное толкование фрагмента речи; представьте себе, как изменяется смысл фразы “I'm looking at the diamond”, когда ее произносит ювелир (“Я разглядываю бриллиант”) или игрок в бейсбол (“Я смотрю на площадку для игры в бейсбол”). В разделе 22.7 будет показано, что прагматическая интерпретация позволяет толковать словосочетание “It is dead” как означающее, что вампир мертв, если мы находимся в той ситуации, в которой существование вампира для нас многое значит.
- **↗ Устранение неоднозначности.** Получатель  $H$  приходит к выводу, что отправитель  $S$  намеревался донести до него смысл высказывания  $P_i$  (где в идеале  $P_i = P$ ). Большинство отправителей сообщений не намерены придавать неоднозначный смысл своим речам, но большинство фрагментов речи имеет несколько допустимых интерпретаций. Тем не менее общение возможно благодаря тому, что получатель берет на себя труд по выяснению того, какой именно является интерпретация, которую отправитель по всей вероятности хотел до него донести. Обратите внимание на то, что здесь впервые в этой главе авторы употребили слово **вероятность**, поскольку устранение неоднозначности — это основной процесс в общении, в котором интенсивно используется формирование рассуждений в условиях неопределенности. В результате анализа вырабатываются возможные интерпретации; если будет обнаружено больше одной интерпретации, то на этапе устранения неоднозначности должен быть сделан выбор той из них, которая является наилучшей.

- **Усвоение.** Получатель  $H$  решает, что нужно поверить (или не поверить) в истинность высказывания  $P_i$ . Совершенно наивный агент может верить всему, что он слышит, но более развитый агент трактует речевой акт как свидетельство в пользу истинности высказывания  $P_i$ , а не как ее подтверждение.

Совмещение всех этих этапов позволяет создать программу агента, приведенную в листинге 22.1. В данном случае агент действует как ведомый робот, которым может командовать ведущий агент. При каждом акте общения ведомый отвечает на вопрос или выполняет команду, если ведущий выдал эту команду, а также принимает на веру все утверждения, высказанные ведущим. Кроме того, он комментирует (однократно) текущую ситуацию, если не имеет никаких срочных дел, которые должен был бы сделать, а также планирует свои собственные действия, если его оставляют в покое. Типичный диалог между ведущим и ведомым приведен в табл. 22.1.

**Таблица 22.1. Типичный диалог между ведущим и ведомым**

Ведомый робот	Ведущий агент
I feel a breeze. (Я чувствую ветерок.)	Go to [1,2]. (Отправляйся в квадрат [1,2].)
Nothing is here. (Здесь ничего нет.)	Go north. (Иди на север.)
I feel a breeze and I smell a stench and I see a glitter. (Я чувствую ветерок, и я ощущаю неприятный запах, и я вижу блеск.)	Grab the gold. (Хватай золото.)

**Листинг 22.1. Общающийся агент, который принимает команды, вопросы и утверждения.** Этот агент способен также описывать текущее состояние или выполнять “обычные” действия, не связанные с речевым актом, если в данных обстоятельствах не о чем говорить

```
function Naive-Communicating-Agent (percept) returns действие action
    static: KB, база знаний
        state, текущее состояние среды
        action, последнее по времени действие, первоначально пустое

    state ← Update-State(state, action, percept)
    words ← Speech-Part(percept)
    semantics ← Disambiguate(Pragmatics(Semantics(Parse(words))))
    if words = None и действием action не является Say
        then /* Описать состояние */
        return Say(Generate-Description(state))
    else if Type[semantics] = Command then /* Подчиниться команде */
        return Contents[semantics]
    else if Type[semantics] = Question then /* Ответить на вопрос */
        answer ← Ask(KB, semantics)
        return Say(Generate-Description(answer))
    else if Type[semantics] = Statement
        then /* Принять на веру утверждение */
        Tell(KB, Contents[semantics])
    /* Если программа дошла до этой точки, выполнить
       "обычное" действие */
    return First(Planner(KB, state))
```

## 22.2. ФОРМАЛЬНАЯ ГРАММАТИКА ДЛЯ ПОДМНОЖЕСТВА АНГЛИЙСКОГО ЯЗЫКА

В данном разделе будет определена формальная грамматика для небольшого подмножества английского языка, которая пригодна для составления утверждений о мире вампуса. Мы будем называть этот язык  $\mathcal{E}_0$ . В следующих разделах  $\mathcal{E}_0$  будет усовершенствован с той целью, чтобы он стал немного ближе к настоящему английскому языку. Авторы едва ли когда-либо предложат полную грамматику для английского языка, даже только по той причине, что вряд ли найдутся два человека, полностью согласные в том, каким должен быть настоящий английский язык.

### Словарь языка $\mathcal{E}_0$

Вначале определим  $\triangleleft$  **словарь**, или список допустимых слов. Эти слова группируются по категориям или частям речи, которые знакомы всем пользователям толковых словарей: существительные, местоимения и имена собственные обозначают предметы, глаголы обозначают события, прилагательные модифицируют существительные, а наречия модифицируют глаголы. Категориями, которые, возможно, менее знакомы некоторым читателям, являются артикли (такие как “the”), предлоги (например, “in”) и союзы (наподобие “and”). В листинге 22.2 приведен небольшой словарь.

#### Листинг 22.2. Словарь языка $\mathcal{E}_0$

---

<i>Noun</i> →	<b>stench</b>	<b>breeze</b>	<b>glitter</b>	<b>nothing</b>	<b>agent</b>	
		<b>wumpus</b>	<b>pit</b>	<b>pits</b>	<b>gold</b>	<b>east</b>
						...
<i>Verb</i> →	<b>is</b>	<b>see</b>	<b>smell</b>	<b>shoot</b>	<b>feel</b>	<b>stinks</b>
		<b>go</b>	<b>grab</b>	<b>carry</b>	<b>kill</b>	<b>turn</b>
						...
<i>Adjective</i> →	<b>right</b>	<b>left</b>	<b>east</b>	<b>dead</b>	<b>back</b>	<b>smelly</b>
						...
<i>Adverb</i> →	<b>here</b>	<b>there</b>	<b>nearby</b>	<b>ahead</b>		
		<b>right</b>	<b>left</b>	<b>east</b>	<b>south</b>	<b>back</b>
						...
<i>Pronoun</i> →	<b>me</b>	<b>you</b>	<b>I</b>	<b>it</b>		...
<i>Name</i> →	<b>John</b>	<b>Mary</b>	<b>Boston</b>	<b>Aristotle</b>		...
<i>Article</i> →	<b>the</b>	<b>a</b>	<b>an</b>			...
<i>Preposition</i> →	<b>to</b>	<b>in</b>	<b>on</b>	<b>near</b>		...
<i>Conjunction</i> →	<b>and</b>	<b>or</b>	<b>but</b>			...
<i>Digit</i> →	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
						<b>6</b>
						<b>7</b>
						<b>8</b>
						<b>9</b>

---

Почти каждая из категорий оканчивается многоточием (...) для указания на то, что в этой категории есть и другие слова. Но следует отметить, что слова показаны не все по двум различным причинам. Что касается существительных, глаголов, прилагательных и наречий, то перечислить их полностью невозможно в принципе. Дело в том, что не только существуют десятки тысяч элементов каждого класса, но и постоянно добавляются все новые и новые, например, такие, как “MP3” или “anime”. Эти четыре категории называются  $\triangleleft$  **открытыми классами**. Другие категории (местоимения, артикли, предлоги и союзы) называются  $\triangleleft$  **закрытыми классами**. Они включают небольшое количество слов (от нескольких слов до нескольких десятков слов), которые можно

в принципе перечислить полностью. Состав закрытых классов изменяется на протяжении столетий, а не месяцев. Например, местоимения “thee” и “thou” широко использовались в английском языке в XVII веке, затем в XIX веке их частота пошла на убыль, а в наши дни они встречаются только в поэзии и в некоторых диалектах.

## Грамматика языка $\mathcal{E}_0$

На следующем этапе необходимо обеспечить объединение слов в словосочетания. Мы будем использовать пять нетерминальных символов для определения словосочетаний различных типов: предложение (Sentence —  $S$ ), именное словосочетание (Noun Phrase —  $NP$ ), глагольное словосочетание (Verb Phrase —  $VP$ ), предложное словосочетание (Prepositional Phrase —  $PP$ ) и относительное предложение<sup>1</sup> (Relative Clause —  $RelClause$ ). Грамматика языка  $\mathcal{E}_0$  приведена в листинге 22.3, где для каждого правила подстановки показан пример. Грамматика  $\mathcal{E}_0$  вырабатывает допустимые английские предложения, например, такие, как показаны ниже.

```
John is in the pit
The wumpus that stinks is in [2,2]
Mary is in Boston and John stinks
```

**Листинг 22.3. Грамматика языка  $\mathcal{E}_0$  с примерами словосочетаний, иллюстрирующих каждое правило**

$S \rightarrow NP\ VP$	I + feel a breeze
$S\ Conjunction\ S$	I feel a breeze + and +
	I smell a wumpus
$NP \rightarrow Pronoun$	I
Name	John
Noun	pits
Article Noun	the + wumpus
Digit Digit	3 4
$NP\ PP$	the wumpus + to the east
$NP\ RelClause$	the wumpus + that is smelly
$VP \rightarrow Verb$	stinks
$VP\ NP$	feel + a breeze
$VP\ Adjective$	is + smelly
$VP\ PP$	turn + to the east
$VP\ Adverb$	go + ahead
$PP \rightarrow Preposition\ NP$	to + the east
$RelClause \rightarrow \text{that } VP$	that + is smelly

К сожалению, эта грамматика не только производит приемлемые предложения, но и допускает ~~перепроизводство~~, т.е. производит предложения, которые не являются грамотными, такие как “Me go Boston” и “I smell pit gold wumpus nothing east”. Кроме того, эта грамматика допускает ~~недопроизводство~~ — она отвергает многие

<sup>1</sup> Относительное предложение следует за именным словосочетанием и модифицирует его. Относительное предложение состоит из относительного местоимения (такого как “who” (кто) или “that” (что)), за которым следует глагольное словосочетание (еще одна разновидность относительного предложения рассматривается в упр. 22.12). Примером относительного предложения является конструкция “that stinks” в предложении “The wumpus that stinks is in [2,2]” (Вампус, который испускает неприятный запах, находится в квадрате [2,2]).

правильные английские предложения, такие как “I think the wumpus is smelly”. (Еще одним недостатком этой грамматики является то, что она не обеспечивает запись первого слова предложения с прописной буквы или добавление точки в конце. Это связано с тем, что данная грамматика предназначена в основном для устной, а не письменной речи.)

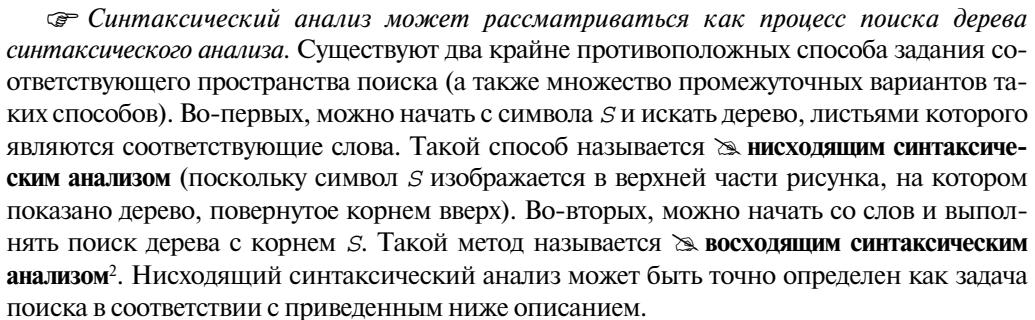
## 22.3. СИНТАКСИЧЕСКИЙ АНАЛИЗ (СИНТАКСИЧЕСКИЙ РАЗБОР)

Выше в этой главе уже было дано определение **синтаксического анализа** как процесса поиска дерева синтаксического анализа для данной конкретной входной строки. Это означает, что вызов функции синтаксического анализа `Parse`, такой как

```
Parse("the wumpus is dead", E0, S)
```

должен привести к получению дерева синтаксического анализа с корнем  $S$ , листьями которого являются слова “the wumpus is dead”, а внутренними узлами — нетерминальные символы грамматики  $E_0$ . Такое дерево показано на рис. 22.1. В виде линейного текста это дерево может быть записано следующим образом:

```
[S: [NP: [Article: the] [Noun: wumpus]]  
[VP: [Verb: is] [Adjective: dead]]]
```

 *Синтаксический анализ может рассматриваться как процесс поиска дерева синтаксического анализа.* Существуют два крайне противоположных способа задания соответствующего пространства поиска (а также множество промежуточных вариантов таких способов). Во-первых, можно начать с символа  $S$  искать дерево, листьями которого являются соответствующие слова. Такой способ называется **нисходящим синтаксическим анализом** (поскольку символ  $S$  изображается в верхней части рисунка, на котором показано дерево, повернутое корнем вверх). Во-вторых, можно начать со слов и выполнять поиск дерева с корнем  $S$ . Такой метод называется **восходящим синтаксическим анализом**<sup>2</sup>. Нисходящий синтаксический анализ может быть точно определен как задача поиска в соответствии с приведенным ниже описанием.

- **Начальное состояние** представляет собой дерево синтаксического анализа, состоящее из корня  $S$  и неизвестных дочерних узлов:  $[S: ?]$ . Вообще говоря, каждое состояние в пространстве поиска также представляет собой дерево синтаксического анализа.
- **Функция определения преемника** выбирает в дереве самый левый узел с неизвестными дочерними узлами. Затем в грамматике осуществляется поиск правил, которые имеют корневую метку узла, находящегося в левой части. Для каждого такого правила создается состояние-преемник, в котором символ  $?$  заменяется списком, соответствующим правой части правила. Например, в грамматике  $E_0$  имеются два правила для  $S$ , поэтому дерево  $[S: ?]$  должно быть заменено следующими двумя преемниками:

<sup>2</sup> Читатель может заметить, что нисходящий и восходящий синтаксический анализ аналогичен прямому и обратному формированию логических рассуждений соответственно, как описано в главе 7. Вскоре будет показано, что эта аналогия является полной.

```
[S: [S: ?] [Conjunction: ?] [S: ?]]  
[S: [NP: ?] [VP: ?]]
```

Второй из этих преемников имеет семь преемников, по одному для каждого правила подстановки *NP*.

- В **проверке цели** проверяется, какие листья дерева синтаксического анализа точно соответствуют входной строке, без неизвестных и неохваченных входных данных.

Одна существенная проблема при нисходящем синтаксическом анализе возникает, когда приходится сталкиваться с так называемыми **леворекурсивными правилами**, т.е. правилами в форме  $X \rightarrow X \dots$ . При поиске в глубину применение такого правила может привести к тому, что замена  $X$  на  $[X: \dots]$  будет осуществляться в бесконечном цикле. А при поиске в ширину можно будет успешно найти варианты синтаксического анализа для допустимых предложений, но при наличии недопустимого предложения может возникнуть такая ситуация, что программа не сможет выйти из бесконечного пространства поиска.

Ниже приведено описание восходящего синтаксического анализа как задачи поиска.

- **Начальным состоянием** является список слов во входной строке, где каждое из слов рассматривается как дерево синтаксического анализа только с одним листовым узлом, например **[the, wumpus, is, dead]**. Вообще говоря, каждое состояние в пространстве поиска представляет собой список деревьев синтаксического анализа.
- С помощью **функции определения преемника** выполняется поиск в каждой позиции  $i$  списка деревьев и в каждой правой части правила грамматики. Если подпоследовательность списка деревьев, начинающаяся с  $i$ , согласуется с правой частью, то эта подпоследовательность заменяется новым деревом, категорией которого является левая часть правила, а дочерними узлами — эта подпоследовательность. Под “согласованием” подразумевается, что категория узла является такой же, как и элемент в правой части. Например, правило *Article* → **the** согласуется с подпоследовательностью, состоящей из первого узла в списке **[the, wumpus, is, dead]**, поэтому состоянием-преемником становится **[Article: the, wumpus, is, dead]**.
- В **проверке цели** проверяется наличие состояния, представляющего собой единственное дерево с корнем *S*.

Пример восходящего синтаксического анализа приведен в табл. 22.2.

И нисходящий, и восходящий синтаксический анализ может оказаться неэффективным из-за того, что отдельные этапы синтаксического анализа различных сочетаний могут комбинироваться самыми разными способами. И в той и в другой процедуре могут возникать непроизводительные затраты времени, связанные с поиском в тех частях пространства состояний, которые не позволяют получить требуемый результат. При нисходящем синтаксическом анализе иногда создаются промежуточные узлы, которые так и не удастся связать со словами, а при восходящем синтаксическом анализе создаются частичные фрагменты синтаксического анализа слов, которые невозможно преобразовать в корневой узел *S*.

Таблица 22.2. Трассировка восходящего синтаксического анализа строки “The wumpus is dead”. Работа начинается со списка узлов, состоящего из отдельных слов. После этого происходит замена подпоследовательностей, соответствующих правой части правила, новым узлом, корнем которого является левая часть правила. Например, в третьей строке показано, как узлы **Article** и **Noun** заменяются узлом **NP**, для которого эти два узла являются дочерними. Нисходящий синтаксический анализ приводит к выработке аналогичной трассировки, но в противоположном направлении

Этап	Список узлов	Подпоследовательность	Правило
Init	<b>the wumpus is dead</b>	<b>the</b>	<b>Article</b> → <b>the</b>
2	<b>Article wumpus is dead</b>	<b>wumpus</b>	<b>Noun</b> → <b>wumpus</b>
3	<b>Article Noun is dead</b>	<b>Article Noun</b>	<b>NP</b> → <b>Article Noun</b>
4	<b>NP is dead</b>	<b>is</b>	<b>Verb</b> → <b>is</b>
5	<b>NP Verb dead</b>	<b>dead</b>	<b>Adjective</b> → <b>dead</b>
6	<b>NP Verb Adjective</b>	<b>Verb</b>	<b>VP</b> → <b>Verb</b>
7	<b>NP VP Adjective</b>	<b>VP Adjective</b>	<b>VP</b> → <b>VP Adjective</b>
8	<b>NP VP</b>	<b>NP VP</b>	<b>S</b> → <b>NP VP</b>
Goal	<b>S</b>		

Но даже если бы существовала идеальная эвристическая функция, позволяющая осуществлять поиск без ненужных отступлений, эти алгоритмы все равно были бы неэффективными, поскольку для некоторых предложений количество деревьев синтаксического анализа измеряется экспоненциальной зависимостью. В следующем подразделе показано, как найти выход из этой ситуации.

## Эффективный синтаксический анализ

Рассмотрим два приведенных ниже предложения.

Have the students in section 2 of Computer Science 101 take the exam.  
Have the students in section 2 of Computer Science 101 taken the exam?

Даже несмотря на то, что первые 10 слов в этих предложениях совпадают, они имеют очень сильно отличающиеся друг от друга деревья синтаксического анализа, поскольку первое из них представляет собой команду, а второе — вопрос. При использовании алгоритма синтаксического анализа, действующего слева направо, пришлось бы принять предположение о том, является ли первое слово частью предложения, представляющего собой команду или вопрос, но нельзя было бы подтвердить правильность этого предположения по меньшей мере до обработки одиннадцатого слова, “take” или “taken”. Если предположение, принятое в этом алгоритме, оказывается неправильным, то приходится осуществлять полный возврат вплоть до первого слова. Такого рода возврат является неизбежным, но для повышения эффективности алгоритма синтаксического анализа следует предотвратить повторный анализ словосочетания “the students in section 2 of Computer Science 101” как *NP* после каждого возврата.

В этом разделе будет разработан алгоритм синтаксического анализа, в котором исключен указанный источник неэффективности. В его основе лежит идея, представляющая собой пример **динамического программирования** — *после проведения анализа каждой подстроки сохранять результаты, чтобы не нужно было проводить ее снова*

*повторный анализ в дальнейшем.* Например, обнаружив, что словосочетание “the students in section 2 of Computer Science 101” относится к типу  $NP$ , можно зарегистрировать этот результат в структуре данных, известной под названием **диаграммы**. Алгоритмы, поддерживающие эти функции, называются **диаграммными синтаксическими анализаторами**. Поскольку в данном случае мы имеем дело с контекстно-свободными грамматиками, то любое словосочетание, обнаруженное в контексте одной ветви пространства поиска, вполне может применяться и в любой другой ветви пространства поиска.

Диаграмма для последовательности из  $n$  строк включает в себя  $n+1$  **вершину** и целый ряд **ребер**, которые соединяют вершины. На рис. 22.2 показана диаграмма с шестью вершинами (обозначенными кружками) и тремя ребрами (обозначенными линиями). Например, ребро со следующей меткой:

$[0, 5, S \rightarrow NP VP \bullet]$

означает, что словосочетание  $NP$ , за которым следует  $VP$ , объединяются, составляя узел  $S$ , который охватывает строку от вершины 0 до вершины 5. Символ  $\bullet$  в обозначении ребра<sup>3</sup> отделяет то, что было найдено до тех пор, от того, что осталось найти. Ребра с символами  $\bullet$  в конце называются **полными ребрами**. Например, следующее ребро:

$[0, 2, S \rightarrow NP \bullet VP]$

указывает, что словосочетание  $NP$  охватывает строку от вершины 0 до вершины 2 (первые два слова) и что если бы мы смогли найти словосочетание  $VP$ , чтобы продолжить это ребро, то получили бы словосочетание  $S$ . Ребра, подобные этому, в которых точка не достигла конца, называются **неполными ребрами**, и принято говорить, что для этого ребра требуется найти словосочетание  $VP$ .

$[0, 5 S \rightarrow NP VP \bullet]$

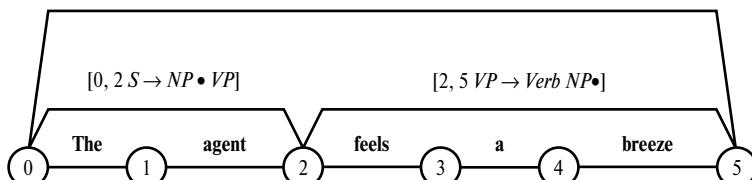


Рис. 22.2. Часть диаграммы, соответствующей предложению “The agent feels a breeze” (Агент чувствует ветерок). Показаны все шесть вершин, но только три из тех ребер, которые требуются для полного синтаксического анализа

Алгоритм диаграммного синтаксического анализатора показан в листинге 22.4. Его основная идея такова, что в нем объединяются лучшие свойства нисходящего и восходящего синтаксического анализа. Процедура `Predictor` выполняет нисходящий анализ; она создает записи в диаграмме, которые указывают, какие символы желательно найти и в каких местах. Процедура `Scanner` — это процедура восходящего анализа, которая начинает работу со слов, но использует любое слово только для продления существующей записи в диаграмме. По аналогии с ней процедура

<sup>3</sup> Именно в связи с тем, что в обозначении ребер применяется символ  $\bullet$ , такие обозначения иногда называют **правилами с точкой**.

Extender формирует составляющие дерева синтаксического анализа в направлении снизу вверх, но только для продления существующей записи в диаграмме.

**Листинг 22.4. Алгоритм диаграммного синтаксического анализатора**, где  $S$  — начальный символ,  $S'$  — новый нетерминальный символ, а  $chart[j]$  — список ребер, которые оканчиваются в вершине  $j$ . Словосочетания, обозначенные греческими буквами, согласуются с некоторой строкой, имеющей длину от нуля и больше символов

---

```

function Chart-Parse(words, grammar) returns диаграмма chart

    chart ← массив[0...Length(words)] пустых списков
    Add-Edge([0, 0, S' → • S])
    for i ← from 0 to Length(words) do
        Scanner(i, words[i])
    return chart

procedure Add-Edge(edge)
    /* Добавить ребро к диаграмме и проверить, позволяет ли оно
       продлить или предсказать другое ребро */
    if ребро edge не находится в диаграмме chart[End(edge)] then
        добавить ребро edge к диаграмме chart[End(edge)]
        if ребро edge не содержит ничего после точки
            then Extender(edge)
        else Predictor(edge)

procedure Scanner(j, word)
    /* Для каждого ребра, ожидающего в данный момент появления слова
       заданной категории, выполнить продление этого ребра */
    for each [i, j, A → α • B β] in chart[j] do
        if слово word относится к категории B then
            Add-Edge([i, j+1, A → α B • β])

procedure Predictor([i, j, A → α • B β])
    /* Добавить к диаграмме любые правила для B, которые могут
       помочь продлить это ребро */
    for each (B → γ) in Rewrites-For(B, grammar) do
        Add-Edge([j, j, B → • γ])

procedure Extender([j, k, B → γ •])
    /* Проверить, какие ребра могут быть продлены с помощью
       этого ребра */
    eb ← ребро, которое является входным для этой процедуры
    for each [i, j, A → α • B' β] in chart[j] do
        if B = B' then
            Add-Edge([i, k, A → α eb • β])

```

---

Для запуска всего алгоритма воспользуемся таким приемом: добавим к диаграмме ребро  $[0, 0, S' \rightarrow • S]$ , где  $S$  — начальный символ грамматики, а  $S'$  — новый символ, который был только что предложен. Вызов процедуры Add-Edge вынуждает процедуру Predictor ввести ребра, соответствующие правилам, применение которых может привести к получению  $S$ , т.е.  $[S \rightarrow NP VP]$ . Затем осуществляется

поиск первой составляющей этого правила,  $NP$ , и добавление правил, соответствующих каждому способу получения какого-то подходящего словосочетания  $NP$ . В конечном итоге процедура предсказателя *Predictor* добавляет в режиме нисходящего синтаксического анализа все возможные ребра, которые могут использоваться в процессе создания окончательного словосочетания  $S$ .

После завершения работы предсказателя применительно к словосочетанию  $S'$  алгоритм входит в цикл, в котором вызывается процедура *Scanner* для каждого слова в предложении. Если слово, находящееся в позиции  $j$ , является членом такой категории  $B$ , которая отыскивается для некоторого ребра в позиции  $j$ , то данное ребро продлевается, а слово отмечается как экземпляр категории  $B$ . Обратите внимание на то, что каждый вызов процедуры *Scanner* приводит к рекурсивному вызову процедур *Predictor* и *Extender*, в результате чего происходит чередование нисходящей и восходящей обработки.

Другой компонент восходящей обработки<sup>4</sup>, процедура *Extender*, принимает на входе некоторое полное ребро с левой частью  $B$  и использует его для продления любого неполного правила в диаграмме, которая оканчивается там, где начинается полное ребро, если для этого неполного правила требуется категория  $B$ .

На рис. 22.3 и в табл. 22.3 показаны диаграмма и трассировка процесса синтаксического анализа с помощью этого алгоритма предложения “I feel it” (которое является положительным ответом на вопрос “Do you feel a breeze?” — “Вы чувствуете ветерок?”). В этой диаграмме зарегистрированы тридцать ребер (обозначенных метками от  $a$  до  $m$ ), включая пять полных ребер (показанных над вершинами диаграммы) и восемь неполных ребер (показанных под вершинами). Обратите внимание на то, как действуют в цикле процедуры предсказания *Predictor*, сканирования *Scanner* и продления *Extender*. Например, в процедуре предсказания используется тот факт, что для ребра  $a$  должен быть выполнен поиск словосочетания  $S$ , чтобы можно было разрешить сделать предсказание о том, что должно появиться словосочетание  $NP$  (ребро  $b$ ), а затем словосочетание *Pronoun* (ребро  $c$ ). После этого процедура сканирования распознает, что в нужном месте находится словосочетание *Pronoun* (ребро  $d$ ), а процедура продления объединяет неполное ребро  $b$  с полным ребром  $d$  для получения нового ребра,  $e$ .

<sup>4</sup> По традиции применяемая нами процедура *Extender* именуется *Completer*. Но последнее название может сбить с толку, поскольку эта процедура не пополняет ребра — она принимает на входе полное ребро и продлевает неполные ребра.

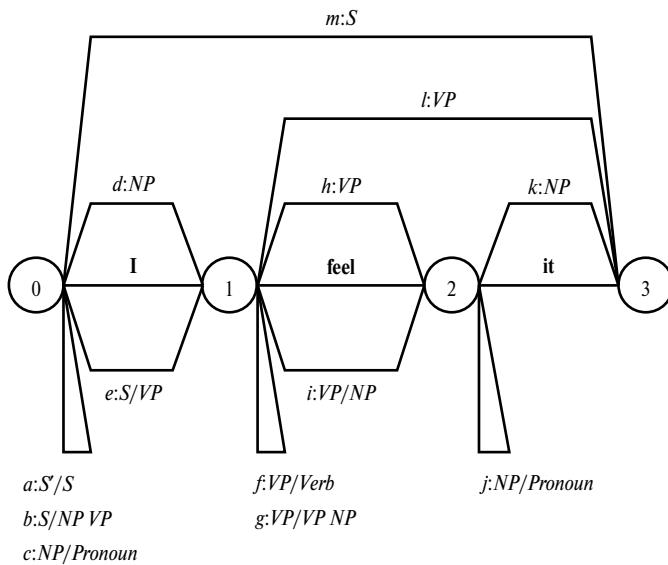


Рис. 22.3. Диаграмма синтаксического анализа предложения “<sub>0</sub> **I** <sub>1</sub> **feel** <sub>2</sub> **it** <sub>3</sub>”. Запись *m:S* означает, что ребро *m* имеет словосочетание *S* в левой части правила, а запись *f:VP/Verb* означает, что ребро *f* имеет словосочетание *VP* в левой части, но для него требуется также выполнить поиск словосочетания *Verb*. В этой диаграмме имеется пять полных ребер, показанных над вершинами, и восемь неполных ребер, показанных под вершинами

Таблица 22.3. Трассировка процесса синтаксического анализа предложения “<sub>0</sub> **I** <sub>1</sub> **feel** <sub>2</sub> **it** <sub>3</sub>”. Для каждого ребра от *a* до *m* показано, какая процедура используется для вывода этого ребра из других ребер, уже имеющихся в диаграмме. Некоторые ребра были исключены в целях сокращения объема таблицы

Ребро	Процедура	Этап вывода
<i>a</i>	Initializer	([0,0, <i>S</i> ' $\rightarrow$ • <i>S</i> ])
<i>b</i>	Predictor( <i>a</i> )	([0,0, <i>S</i> $\rightarrow$ • <i>NP VP</i> ])
<i>c</i>	Predictor( <i>b</i> )	([0,0, <i>NP</i> $\rightarrow$ • <i>Pronoun</i> ])
<i>d</i>	Scanner( <i>c</i> )	([0,1, <i>NP</i> $\rightarrow$ <i>Pronoun</i> •])
<i>e</i>	Extender( <i>b,d</i> )	([0,1, <i>S</i> $\rightarrow$ <i>NP</i> • <i>VP</i> ])
<i>f</i>	Predictor( <i>e</i> )	([1,1, <i>VP</i> $\rightarrow$ • <i>Verb</i> ])
<i>g</i>	Predictor( <i>e</i> )	([1,1, <i>VP</i> $\rightarrow$ • <i>VP NP</i> ])
<i>h</i>	Scanner( <i>f</i> )	([1,2, <i>VP</i> $\rightarrow$ <i>Verb</i> •])
<i>i</i>	Extender( <i>g,h</i> )	([1,2, <i>VP</i> $\rightarrow$ <i>VP</i> • <i>NP</i> ])
<i>j</i>	Predictor( <i>g</i> )	([2,2, <i>NP</i> $\rightarrow$ • <i>Pronoun</i> ])
<i>k</i>	Scanner( <i>j</i> )	([2,3, <i>NP</i> $\rightarrow$ <i>Pronoun</i> •])
<i>l</i>	Extender( <i>i,k</i> )	([1,3, <i>VP</i> $\rightarrow$ <i>VP NP</i> •])
<i>m</i>	Extender( <i>e,l</i> )	([0,3, <i>S</i> $\rightarrow$ <i>NP VP</i> •])

Этот алгоритм диаграммного синтаксического анализатора позволяет предотвратить формирование большого класса ребер, которые пришлось бы исследовать при использовании простой восходящей процедуры. Рассмотрим предложение: “The ride the horse gave was wild” (Скачка, на которую пустилась эта лошадь, была дикой). Процедура восходящего синтаксического анализа отметила бы словосочетание “ride the horse” (скакать на лошади) как  $VP$ , а затем отбросила бы все дерево синтаксического анализа, обнаружив, что такой вариант анализа не складывается в общее словосочетание  $S$ . Но в языке  $\mathcal{E}_0$  не разрешается, чтобы словосочетание  $VP$  следовало за словом “the”, поэтому данный алгоритм диаграммного синтаксического анализатора никогда не предсказал бы наличие в этой точке словосочетания  $VP$  и поэтому избежал бы напрасного расходования времени на формирование здесь составляющей  $VP$ . Алгоритмы, которые действуют слева направо и предотвращают формирование таких невозможных составляющих, называются синтаксическими анализаторами по левому углу, поскольку они формируют дерево синтаксического анализа, начинаяющееся с начального символа грамматики и продолжающееся вниз к самому левому слову предложения (к левому углу). Ребро добавляется к диаграмме, только если оно может послужить для продления этого дерева синтаксического анализа (соответствующий пример приведен на рис. 22.4).

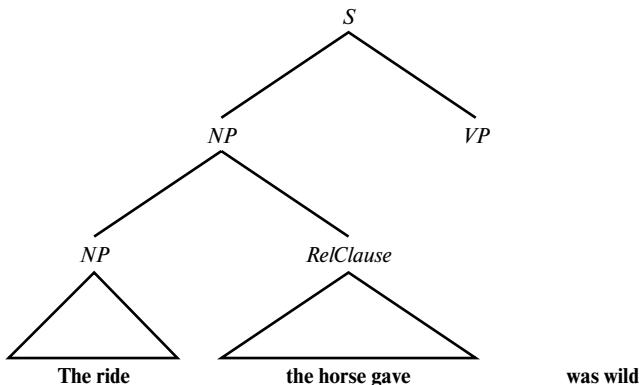


Рис. 22.4. Алгоритм синтаксического анализа по левому углу позволяет предотвратить предсказание словосочетания  $VP$ , начинающееся со слова “ride”, но предсказывает словосочетание  $VP$ , начинающееся со слова “was”, поскольку в данной грамматике допускается наличие словосочетания  $VP$ , которое следует за  $NP$ . Треугольник, показанный над словами “the horse gave” (на которую пустилась эта лошадь), означает, что для этих слов был сделан синтаксический анализ, который показал, что они представляют собой относительное предложение  $RelClause$ , но дополнительные промежуточные составляющие этого синтаксического анализа не показаны

Этот диаграммный синтаксический анализатор характеризуется только полиномиальными затратами времени и пространства. Он требует  $O(kn^2)$  пространства для хранения ребер, где  $n$  — количество слов в предложении, а  $k$  — константа, которая зависит от грамматики. Если алгоритм не может продолжать дальнейшее формирование ребер, он останавливается, поэтому известно, что работа данного алгоритма всегда завершается (даже при наличии леворекурсивных правил). В действительности

сти он требует времени  $O(n^3)$  даже в наихудшем случае, а это — самые лучшие показатели, которые могут быть достигнуты при использовании контекстно-свободных грамматик. Узким местом алгоритма Chart-Parse является процедура Extender, которая должна предпринять попытку продления каждого из  $O(n)$  неполных ребер, оканчивающихся в позиции  $j$ , с помощью каждого из  $O(n)$  полных ребер, начинающихся с позиции  $j$ , для каждого из  $n+1$  различных значений  $j$ . Перемножая эти числа, получаем значение  $O(n^3)$ . Полученные результаты довольно парадоксальны — как может алгоритм с затратами времени  $O(n^3)$  возвратить ответ, в котором может содержаться экспоненциальное количество деревьев синтаксического анализа? Рассмотрим один пример; следующее предложение:

Fall leaves fall and spring leaves spring

является неоднозначным, поскольку в нем каждое слово (кроме “and”) может быть либо существительным, либо глаголом, а “fall” и “spring” могут быть также прилагательными. В целом этому предложению могут соответствовать четыре приведенных ниже варианта<sup>5</sup> синтаксического анализа.

```
[S: [S: [NP: Fall leaves] fall] and [S: [NP: spring leaves] spring]
[S: [S: [NP: Fall leaves] fall] and [S: spring [VP: leaves spring]]
[S: [S: Fall [VP: leaves fall]] and [S: [NP: spring leaves] spring]
[S: [S: Fall [VP: leaves fall]] and [S: spring [VP: leaves spring]]]
```

Если имеется  $n$  неоднозначных сочетающихся друг с другом фрагментов предложения, то может быть предусмотрено  $2^n$  способов<sup>6</sup> выбора вариантов синтаксического анализа для этих фрагментов предложения. Как избежать экспоненциального роста затрат времени на обработку при использовании данного алгоритма диаграммного синтаксического анализатора? По сути на этот вопрос можно дать два ответа. Первый ответ состоит в том, что сам алгоритм Chart-Parse фактически представляет собой не синтаксический анализатор, а распознаватель. Если в диаграмме имеется полное ребро в форме  $[0, n, S \rightarrow \alpha \bullet]$ , это означает, что какое-то словосочетание  $S$  распознано. Восстановление дерева синтаксического анализа из информации, заданной в этом ребре, не рассматривается как часть функций алгоритма Chart-Parse, но может быть выполнено с его помощью. Обратите внимание на то, что в последней операции процедуры Extender строка  $\alpha$  формируется как список ребер  $e_B$ , а не просто как список имен категорий. Поэтому, чтобы преобразовать ребро в дерево синтаксического анализа, достаточно выполнить рекурсивный просмотр составляющих ребер, преобразовав каждое ребро  $[i, j, X \rightarrow \alpha \bullet]$  в дерево  $[X: \alpha]$ . Такой процесс является несложным, но позволяет получить только одно дерево синтаксического анализа.

Второй ответ состоит в том, что если требуются все возможные варианты синтаксического анализа, то придется глубже разобраться в этой диаграмме. Во время преобразования ребра  $[i, j, X \rightarrow \alpha \bullet]$  в дерево  $[X: \alpha]$  можно будет проверить, есть ли какие-либо другие ребра в форме  $[i, j, X \rightarrow \beta \bullet]$ . Если они имеются, то эти

<sup>5</sup> Вариант синтаксического анализа  $[S: \text{Fall} [\text{VP: leaves fall}]]$  эквивалентен предложению “Autumn abandons autumn” (Осень покидает осень).

<sup>6</sup> Может возникнуть также  $O(n!)$  неоднозначных вариантов, связанных с тем способом, как компоненты соединяются друг с другом, например  $(X \text{ и } Y) \text{ и } Z$  или  $((X \text{ и } Y) \text{ и } Z)$ . Но это — другая проблема, которая весьма успешно исследована в [256].

ребра должны сформировать дополнительные варианты синтаксического анализа. Определив все эти варианты, необходимо будет продумать вопрос, что с ними делать. Можно просто перечислить все варианты, а это означает, что указанный выше парадокс будет разрешен и потребуется экспоненциальное количество времени для составления списка всех вариантов синтаксического анализа. Еще один подход может состоять в том, чтобы еще немного продолжить эту работу и представить варианты синтаксического анализа в виде структуры, называемой **упакованным лесом**, который выглядит примерно так:

```
[S: [S: { [NP: Fall leaves] [VP: fall] } ] and
  [NP: Fall] [VP: leaves fall]
  [S: { [NP: spring leaves] [VP: spring] } ]
  [NP: spring] [VP: leaves spring]]]
```

Идея этого способа представления состоит в том, что каждый узел дерева синтаксического анализа может быть либо обычным узлом дерева, либо множеством узлов дерева. Это позволяет возвратить из программы некоторое представление экспоненциального количества вариантов синтаксического анализа за счет полиномиальных затрат пространства и времени. Безусловно, если  $n=2$ , то разницы между  $2^n$  и  $2n$  нет, но при больших значениях  $n$  такое представление обеспечивает значительную экономию. К сожалению, этот простой подход на основе упакованного леса не позволяет учесть все  $O(n!)$  вариантов неоднозначности, касающиеся того, как могут быть связаны между собой различные составляющие. Максвелл и Каплан [1002] показали, что некоторый более сложный способ представления, основанный на принципах систем поддержки истинности, позволяет упаковывать деревья еще плотнее.

## 22.4. РАСШИРЕННЫЕ ГРАММАТИКИ

Как было показано в разделе 22.2, простая грамматика для языка  $\mathcal{E}_0$  производит предложение “I smell a stench” (Я чувствую неприятный запах) и много других предложений на английском языке. К сожалению, она производит также много словосочетаний, не являющихся предложениями, таких как “Me smell a stench”. Для предотвращения этой проблемы в этой грамматике нужно было бы учесть, что слово “me” не является допустимым словосочетанием  $NP$  при его использовании в качестве подлежащего некоторого предложения. Как говорят лингвисты, местоимение “I” находится в именительном падеже, а “me” — в объектном падеже<sup>7</sup>. После того как учтено наличие падежа, становится очевидно, что грамматика  $\mathcal{E}_0$  не является контекстно-свободной, поскольку нельзя утверждать, что любое словосочетание  $NP$  эквивалентно любому другому независимо от контекста. Этот недостаток можно исправить, введя новые категории, такие как  $NP_S$  и  $NP_O$ , для обозначения именных словосочетаний в именительном и объектном падеже соответственно. Для этого также потребуется разделить категорию местоимений  $Pronoun$  на две категории —  $Pronoun_S$  (которая включает “I”) и  $Pronoun_O$  (которая включает “me”). В верхней части лис-

<sup>7</sup> Именительный падеж иногда называют также *номинативным падежом*, а объектный падеж — *винительным падежом*. Во многих языках имеется также дательный падеж для слов в позиции косвенного объекта.

тинга 22.5 показана полная грамматика BNF, подготовленная с учетом соглашения о падеже; мы будем называть полученный в результате язык  $\mathcal{E}_1$ . Обратите внимание на то, что пришлось продублировать все правила  $NP$ , введя одно правило для  $NP_S$ , а другое — для  $NP_O$ .

**Листинг 22.5.** Варианты представления грамматики. В верхней части показана грамматика BNF для языка  $\mathcal{E}_1$ , в котором учитываются именительный и объектный падежи в именных словосочетаниях и поэтому не допускается такое существенное перепроизводство, как в языке  $\mathcal{E}_0$ . Части грамматики, идентичные грамматике  $\mathcal{E}_0$ , были исключены. В нижней части показана грамматика определенных выражений (Definite Clause Grammar — DCG) языка  $\mathcal{E}_1$

$S \rightarrow NPS \ VP \mid \dots$
$NPS \rightarrow Pronouns_S \mid Name \mid Noun \mid \dots$
$NP_O \rightarrow Pronoun_O \mid Name \mid Noun \mid \dots$
$VP \rightarrow VP \ NP_O \mid \dots$
$PP \rightarrow Preposition \ NP_O$
$Pronouns_S \rightarrow I \mid you \mid he \mid she \mid it \mid \dots$
$Pronoun_O \rightarrow me \mid you \mid him \mid her \mid it \mid \dots$
$S \rightarrow NP(\text{Subjective}) \ VP \mid \dots$
$NP(case) \rightarrow Pronoun(case) \mid Name \mid Noun \mid \dots$
$VP \rightarrow VP \ NP(\text{Objective}) \mid \dots$
$PP \rightarrow Preposition \ NP(\text{Objective})$
$Pronoun(\text{Subjective}) \rightarrow I \mid you \mid he \mid she \mid it \mid \dots$
$Pronoun(\text{Objective}) \rightarrow me \mid you \mid him \mid her \mid it \mid \dots$

К сожалению, язык  $\mathcal{E}_1$  все еще допускает перепроизводство. В английском и многих других языках требуется ~~согласование~~ согласование между подлежащим и основным глаголом предложения. Например, если подлежащим является “I”, то предложение “I smell” является грамматически правильным, а “I smells” — нет. Если же подлежащим является “it”, то верно противоположное. В английском языке различия, обусловленные согласованием, являются минимальными: большинство глаголов имеют одну форму для подлежащего, представляющего собой местоимение третьего лица в единственном числе (“he”, “she” или “it”), и вторую форму для всех других комбинаций лица и числа. Есть только одно исключение: словосочетания “I am / you are / he is” имеют три формы. Умножая эти три разные формы на две разные формы  $NP_S$  и  $NP_O$ , получим шесть форм  $NP$ . Если бы количество различий было еще больше, то в конечном итоге общее количество вариантов выражалось бы экспоненциальной зависимостью.

Альтернативный подход состоит в том, что необходимо ~~расширить~~ расширить существующие правила грамматики, а не вводить новые правила. Вначале приведем пример того, как могли бы выглядеть расширенные правила (см. нижнюю часть листинга 22.5), а затем формально определим способ интерпретации этих правил. Расширенные правила допускают применение параметров в нетерминальных категориях. В листинге 22.5 показано, как описать грамматику  $\mathcal{E}_1$  с использованием расширенных правил. Категории  $NP$  и  $Pronoun$  имеют параметр, обозначающий их падеж. (Существительные не имеют падежа в английском языке, но имеют его во многих других языках.) В правиле для  $S$  словосочетание  $NP$  должно находиться в именительном падеже, тогда как в правилах для  $VP$  и  $PP$  словосочетание  $NP$  должно

находиться в объектном падеже. Правило для  $NP$  принимает в качестве параметра некоторую переменную,  $case$ . Назначение этой конструкции состоит в том, что словосочетание  $NP$  может иметь любой падеж, но если вместо этого словосочетания  $NP$  подставляется местоимение  $Pronoun$ , то последнее должно иметь такой же падеж. Применение переменных, позволяющее исключить необходимость принятия решения, когда различие, определяемое переменной, не имеет значения, способствует тому, что не приходится увеличивать размер множества правил в экспоненциальной зависимости от количества рассматриваемых характеристик.

Используемая для такого дополнения формальная система называется **GRAMMATIKOЙ ОПРЕДЕЛЕННЫХ ВЫРАЖЕНИЙ**, или сокращенно DCG (Definite Clause Grammar), поскольку каждое правило этой грамматики можно интерпретировать как определенное выражение<sup>8</sup> в хорновской логике. Вначале мы покажем, как можно интерпретировать в качестве определенного выражения обычные, нерасширенные правила. Мы будем рассматривать каждый символ с обозначением категории как предикат, заданный на строках, поэтому выражение  $NP(s)$  принимает истинное значение, если строка  $s$  образует некоторое словосочетание  $NP$ . Следующее правило CFG:

$$S \rightarrow NP \ VP$$

является сокращенным обозначением для следующего определенного выражения:

$$NP(s_1) \wedge VP(s_2) \Rightarrow S(s_1 + s_2)$$

Здесь  $s_1 + s_2$  обозначает конкатенацию двух строк, поэтому данное правило говорит о том, что если строка  $s_1$  представляет собой словосочетание  $NP$ , а строка  $s_2$  — словосочетание  $VP$ , то их конкатенация представляет собой словосочетание  $S$ ; это означает, что данная интерпретация точно соответствует тому, как мы уже интерпретировали это правило CFG. Важно отметить, что **GRAMMATIKOЙ ОПРЕДЕЛЕННЫХ ВЫРАЖЕНИЙ ПОЗВОЛЯЕТ РАССМАТРИВАТЬ СИНТАКСИЧЕСКИЙ АНАЛИЗ КАК ЛОГИЧЕСКИЙ ВЫВОД**. Такой подход открывает возможность рассуждать о языках и строках с помощью многих разных способов. Например, он означает, что восходящий синтаксический анализ можно осуществлять с помощью прямого логического вывода, а нисходящий синтаксический анализ — с помощью обратного логического вывода. Как будет показано ниже, это также означает, что одна и та же грамматика может использоваться и для синтаксического анализа, и для производства новых предложений.

Но самым важным преимуществом подхода на основе DCG является то, что он дает возможность дополнять символы категорий вспомогательными параметрами, отличными от обычных строковых параметров. Например, следующее правило:

$$NP(case) \rightarrow Pronoun(case)$$

является сокращенным обозначением для такого определенного выражения:

$$Pronoun(case, s_1) \Rightarrow NP(case, s_1)$$

Это правило гласит, что если строка  $s_1$  представляет собой местоимение  $Pronoun$  с падежом, указанным переменной  $case$ , то  $s_1$  также представляет собой словосочетание  $NP$  с тем же падежом. Вообще говоря, символ категории может быть

<sup>8</sup> Напомним, что определенное выражение, будучи записанным в виде импликации, имеет в своем следствии точно один атом, а в своей предпосылке — конъюнкцию атомов в количестве от нуля или больше. Двумя примерами таких выражений являются  $A \wedge B \Rightarrow C$  и просто  $C$ .

дополнен любым количеством формальных параметров, а сами формальные параметры могут заменяться фактическими параметрами, подлежащими унификации, как в обычном логическом выводе с использованием хорновских выражений.

Но за такое удобство приходится платить: в распоряжение составителя грамматики передается все мощь системы автоматического доказательства теорем, поэтому приходится отказываться от гарантированного выполнения синтаксического анализа за время  $O(n^3)$ ; задача синтаксического анализа грамматики с расширениями может оказаться NP-трудной или даже неразрешимой, в зависимости от этих расширений.

Для того чтобы обеспечить применение грамматики DCG, необходимо воспользоваться еще несколькими приемами; например, требуется способ задания терминальных символов, кроме того, было бы удобно иметь возможность не вводить автоматический строковый параметр. Учтя все эти пожелания, авторы решили, что нужно задать грамматику определенных выражений как описано ниже.

- Запись  $X \rightarrow Y Z \dots$  должна быть преобразована в запись  $Y(s_1) \wedge Z(s_2) \wedge \dots \Rightarrow X(s_1+s_2+\dots)$ .
- Запись  $X \rightarrow Y | Z | \dots$  должна быть преобразована в запись  $Y(s) \vee Z(s) \vee \dots \Rightarrow X(s)$ .
- В том и в другом приведенном выше правиле любой нетерминальный символ  $Y$  может быть дополнен одним или несколькими параметрами. Каждый параметр может представлять собой переменную, константу или функцию от некоторых параметров. После перевода в форму определенного выражения эти параметры должны предшествовать строковому параметру (например, терм  $NP(case)$  переводится в форму  $NP(case, s_1)$ ).
- Запись  $\{P(\dots)\}$  может появляться в правой части правила и буквально переводится в  $P(\dots)$ . Это позволяет составителю грамматики вставлять проверку для терма  $P(\dots)$  без необходимости добавления автоматического строкового параметра.
- Запись  $X \rightarrow \text{word}$  переводится как  $X([\text{word}])$ .

Проблему согласования подлежащего и глагольного сказуемого также можно решить с помощью дополнений, но мы отложим эту задачу до упр. 22.2. Вместо этого решим более сложную задачу, такую как субкатегоризация глагола.

## Субкатегоризация глагола

Грамматика  $\mathcal{E}_1$  является более совершенной по сравнению  $\mathcal{E}_0$ , но все еще допускает перепроизводство. Один из ее недостатков состоит в том, какой способ применяется для соединения глагольных словосочетаний с другими словосочетаниями. Требуется, чтобы считались приемлемыми глагольными словосочетаниями только такие группы слов, как “give me the gold” (отдай мне золото) и “go to [1,2]” (отправляйся в квадрат [1,2]). Все эти словосочетания входят в состав языка  $\mathcal{E}_1$ , но, к сожалению, порождаются также словосочетания “go me the gold” и “give to [1,2]”. Язык  $\mathcal{E}_2$  позволяет устранить подобные словосочетания  $VP$  путем явного указания на то, какие словосочетания могут следовать за теми или иными глаголами. Список с такой информацией называется списком **субкатегоризации** для глагола. Идея этого подхода состоит в том, что категория  $Verb$  подразделяется на субкатегории —

на ту, в которой глаголы не имеют объекта, в которой глаголы могут иметь один объект, и т.д.

Для реализации этой идеи для каждого глагола предусматривается **список субкатегоризации**, в котором перечислены **дополнения** глагола. Дополнением называется обязательное словосочетание, которое следует за глаголом в глагольном словосочетании. Поэтому в словосочетании “Give the gold to me” словосочетания *NP* “the gold” и *PP* “to me” являются дополнениями<sup>9</sup> глагола “give”. Для записи такого списка применяется следующая форма:

*Verb*( [ *NP*, *PP* ] ) → **give** | **hand** | ...

Допустимо, чтобы глагол имел несколько различных субкатегоризаций, точно так же, как и слово может принадлежать к нескольким разным категориям. В действительности и глагол “give” имеет список субкатегоризации [ *NP*, *NP* ], как в словосочетании “Give me the gold”. Такую ситуацию можно рассматривать как еще одну разновидность неоднозначности, требующей устранения. В табл. 22.4 приведены некоторые примеры глаголов и их списков субкатегоризации (или сокращенно **субкатегоризаций**).

Таблица 22.4. Примеры глаголов с их списками субкатегоризации

Глагол	Список субкатегоризации	Пример глагольного словосочетания
give	[ <i>NP</i> , <i>PP</i> ]	give the gold in 3 to me
	[ <i>NP</i> , <i>NP</i> ]	give me the gold
smell	[ <i>NP</i> ]	smell a wumpus
	[ <i>Adjective</i> ]	smell awful
	[ <i>PP</i> ]	smell like a wumpus
is	[ <i>Adjective</i> ]	is smelly
	[ <i>PP</i> ]	is in 2 2
	[ <i>NP</i> ]	is a pit
died	[]	died
believe	[ <i>S</i> ]	believe the wumpus is dead

Для того чтобы включить механизм субкатегоризации глагола в грамматику, необходимо выполнить три этапа. На первом этапе требуется дополнить категорию *VP* так, чтобы она принимала параметр субкатегоризации, *VP*(*subcat*), который обозначает список дополнений, необходимых для формирования полного словосочетания *VP*. Например, глагол “give” можно преобразовать в полное словосочетание *VP*, добавив субкатегоризацию [ *NP*, *PP* ], словосочетание “give the gold” можно сделать полным, добавив субкатегоризацию [ *PP* ], а словосочетание “give the gold to me” уже является полным словосочетанием *VP*, поэтому его список субкатегоризации представляет собой пустой список, [ ]. На основании этих рассуждений можно составить следующие правила для словосочетания *VP*:

<sup>9</sup> Это — одно из определений понятия дополнения, но другие авторы применяют иную терминологию. Некоторые указывают, что в английском языке подлежащее, с которым связан глагол, также подчиняется дополнению, а другие считают, что дополнением следует называть только предложное словосочетание, а именное словосочетание необходимо рассматривать как **параметр**.

```


$$\begin{aligned} VP(subcat) \rightarrow & Verb(subcat) \\ | & VP(subcat + [NP]) NP(Objective) \\ | & VP(subcat + [Adjective]) Adjective \\ | & VP(subcat + [PP]) PP \end{aligned}$$


```

Последнюю строку этих правил можно прочитать таким образом: “Словосочетание  $VP$  с заданным списком субкатегоризации  $subcat$  можно сформировать на основе вложенного словосочетания  $VP$ , за которым следует  $PP$ , при условии, что вложенное  $VP$  имеет список субкатегоризации, который начинается с элементов списка  $subcat$  и заканчивается символом  $PP$ ”. Например, выражение  $VP([])$  формируется с помощью  $VP([PP])$ , за которым следует  $PP$ . А первая строка этих правил говорит о том, что словосочетание  $VP$  со списком субкатегоризации  $subcat$  может быть сформировано с помощью категории  $Verb$  с тем же самым списком субкатегоризации. В частности, выражение  $VP([NP])$  можно сформировать с помощью  $Verb([NP])$ . Одним из примеров такого глагола является “grab”, поэтому словосочетание “grab the gold” (хватаю золото) обозначается как  $VP([])$ .

На втором этапе необходимо изменить правило для  $S$  так, чтобы в нем требовалось глагольное словосочетание, имеющее при себе все свои дополнения и поэтому сопровождаемое списком субкатегоризации  $[]$ . Это означает, что словосочетание “I grab the gold” (Я хватаю золото) представляет собой допустимое предложение, а “You give” — нет. Таким образом, вводится в действие следующее новое правило:

$$S \rightarrow NP(Subjective) \ VP([])$$

которое можно прочитать так: “Предложение может состоять из словосочетания  $NP$  в именительном падеже, за которым следует словосочетание  $VP$ , имеющее пустой список субкатегоризации”. На рис. 22.5 показано дерево синтаксического анализа, полученное с помощью этой грамматики.

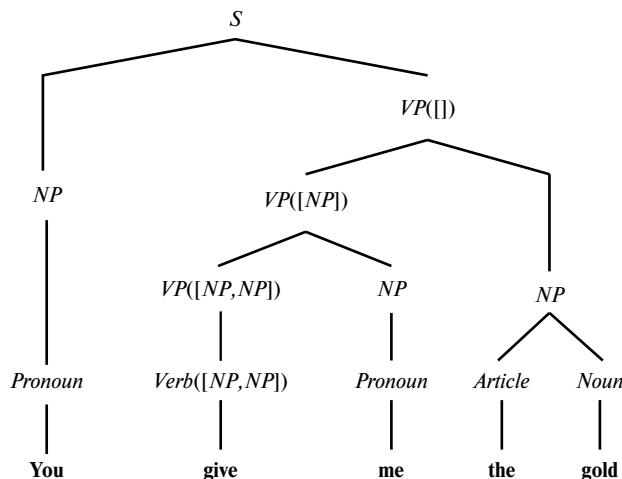


Рис. 22.5. Дерево синтаксического анализа предложения “You give me the gold”, на примере которого показана субкатегоризация глагола и глагольного словосочетания

На третьем этапе должно быть учтено то, что глагольные словосочетания (и другие словосочетания), кроме дополнений, могут также иметь **приложения**, или отглагольные дополнения, представляющие собой словосочетания, которые не связаны с отдельным глаголом, а, скорее, могут появляться в любом глагольном словосочетании. К приложениям относятся словосочетания, представляющие время и место, поскольку почти любое действие или событие может быть связано с определенным временем или местом. Например, приложениями являются наречие “now” в словосочетании “I smell a wumpus now” (Сейчас я чувствую запах вампуса) и предложное словосочетание *PP* “on Tuesday” в предложении “give me the gold on Tuesday” (Отдай мне золото во вторник). Ниже приведены два правила, которые допускают включение предложного и обстоятельственного дополнения в любое глагольное словосочетание *VP*.

$$\begin{aligned} VP(subcat) \rightarrow & VP(subcat) PP \\ | & VP(subcat) Adverb \end{aligned}$$

### Порождающая мощь расширенных грамматик

Каждое расширенное правило представляет собой **схему правила**, заменяющую собой множество правил, по одному для каждой возможной комбинации значений расширенных составляющих. Порождающая способность расширенных грамматик зависит от количества таких комбинаций. Если их количество является конечным, то расширенная грамматика эквивалента контекстно-свободной грамматике: схема правила может быть заменена отдельными контекстно-свободными правилами. Если же количество значений бесконечно, то расширенные грамматики могут представлять языки, отличные от контекстно-свободных. Например, контекстно-зависимый язык  $a^n b^n c^n$  может быть представлен следующим образом:

$$\begin{array}{ll} S(n) \rightarrow A(n) B(n) C(n) & \\ A(1) \rightarrow a & A(n+1) \rightarrow a A(n) \\ B(1) \rightarrow b & B(n+1) \rightarrow b B(n) \\ C(1) \rightarrow c & C(n+1) \rightarrow c C(n) \end{array}$$

## 22.5. СЕМАНТИЧЕСКАЯ ИНТЕРПРЕТАЦИЯ

До сих пор в данной главе рассматривался только синтаксический анализ языка. В этом разделе обратимся к **семантике** — извлечению смысла фрагментов речи. В данной главе в качестве языка представления используется логика первого порядка, поэтому семантическая интерпретация представляет собой процесс связывания с некоторым словосочетанием некоторого выражения логики первого порядка. Интуитивно ясно, что смысл словосочетания “the wumpus” — это образ большого, мохнатого животного, который может быть представлен в логике в виде логического терма *Wumpus*<sub>1</sub>, а смыслом словосочетания “the wumpus is dead” является логическое выражение *Dead(Wumpus*<sub>1</sub>). В этом разделе такие интуитивные представления будут определены более точно. Начнем с простого примера — с правила для описания позиций в решетке:

$$NP \rightarrow Digit\ Digit$$

Дополним данное правило, добавив к каждой составляющей параметр, который представляет семантику этой составляющей. Будет получено следующее:

$$NP([x, y]) \rightarrow Digit(x) \ Digit(y)$$

Это правило гласит, что строка, состоящая из цифры с семантикой  $x$ , за которой следует еще одна цифра с семантикой  $y$ , образует словосочетание  $NP$  с семантикой  $[x, y]$ , которое представляет собой применяемое нами обозначение квадрата в решетке — в мире вампуша.

Обратите внимание на то, что семантика всего словосочетания  $NP$  в основном формируется на основе семантики составляющих частей. Мы уже сталкивались с подобной идеей  $\bowtie$  композиционной семантики и раньше: в логике смысл выражения  $P \wedge Q$  определяется значениями  $P$ ,  $Q$  и  $\wedge$ , а в арифметике смысл выражения  $x+y$  определяется значениями  $x$ ,  $y$  и  $+$ . В листинге 22.6 показано, как можно воспользоваться системой обозначений DCG, чтобы дополнить грамматику арифметических выражений с семантикой, а на рис. 22.6 показано дерево синтаксического анализа выражения  $3 + (4 \div 2)$  согласно этой грамматике. Корнем данного дерева синтаксического анализа является выражение  $Exp(5)$ , семантической интерпретацией которого становится 5.

**Листинг 22.6. Грамматика арифметических выражений, расширенная с учетом семантики. Каждая переменная  $x_i$  представляет семантику одной из составляющих. Обратите внимание на использование обозначения операции проверки  $\{test\}$  для определения логических предикатов, которые должны быть удовлетворены, но не являются составляющими**

---

```

 $Exp(x) \rightarrow Exp(x_1) \ Operator(op) \ Exp(x_2) \ \{x=Apply(op, x_1, x_2)\}$ 
 $Exp(x) \rightarrow (Exp(x))$ 
 $Exp(x) \rightarrow Number(x)$ 
 $Number(x) \rightarrow Digit(x)$ 
 $Number(x) \rightarrow Number(x_1) \ Digit(x_2) \ \{x=10 \times x_1+x_2\}$ 
 $Digit(x) \rightarrow x \ \{0 \leq x \leq 9\}$ 
 $Operator(x) \rightarrow x \ \{x \in \{+, -, \div, \times\}\}$ 

```

---

### Семантика небольшой части английского языка

Теперь мы готовы записать семантические дополнения для небольшой части английского языка. Начнем с определения того, какие семантические представления желательно связать с теми или иными словосочетаниями. Рассмотрим простой пример предложения “John loves Mary” (Джон любит Мэри). Словосочетание  $NP$  “John” должно иметь в качестве его семантической интерпретации логический терм  $John$ , а все предложение в целом должно иметь в качестве своей интерпретации логическое высказывание  $Loves(John, Mary)$ . Эти определения пока не вызывают затруднений. Сложности возникают при анализе глагольного словосочетания  $VP$  “loves Mary”. Семантическая интерпретация этого словосочетания не является ни логическим термом, ни полным логическим высказыванием. Интуитивно ясно, что слова “loves Mary” представляют собой описание, которое может относиться или не относиться к конкретному лицу (в данном случае оно относится к Джону). Это означает, что словосочетание “loves Mary” представляет собой **предикат**, который позволяет получить полное логическое высказывание после его объединения с термом,

представляющим некоторое лицо (то лицо, которое выражает любовь). Используя  $\lambda$ -обозначение (см. с. 351), можно представить словосочетание “*loves Mary*” как следующий предикат:

$$\lambda x \text{ Loves}(x, \text{Mary})$$

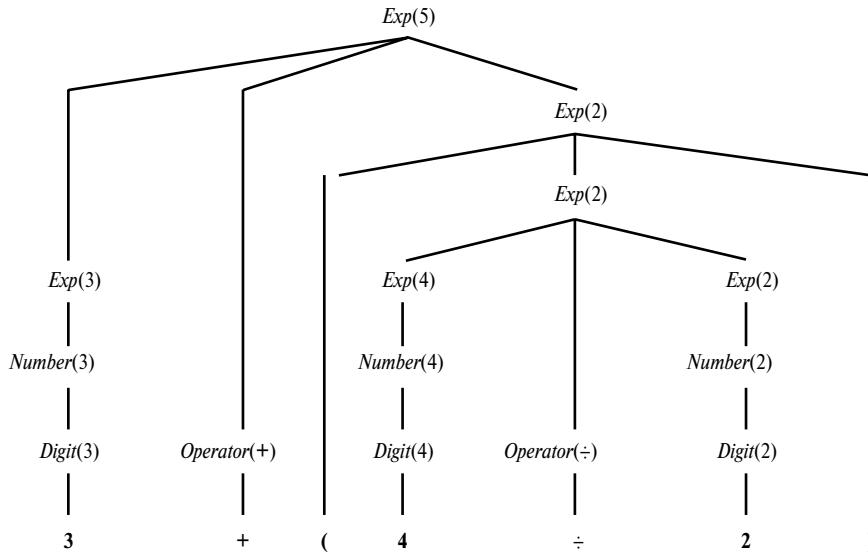


Рис. 22.6. Дерево синтаксического анализа с семантическими интерпретациями для строки “3 + (4 ÷ 2)”

Теперь необходимо сформулировать правило, которое означает следующее: “Словосочетание  $NP$  с семантикой  $obj$ , за которым следует словосочетание  $VP$  с семантикой  $rel$ , составляют высказывание, семантика которого является результатом применения  $rel$  к  $obj$ ”:

$$S(rel(obj)) \rightarrow NP(obj) VP(rel)$$

Это правило сообщает, что семантической интерпретацией предложения “*John loves Mary*” является следующая:

$$(\lambda x \text{ Loves}(x, \text{Mary})) (\text{John})$$

которая эквивалентна выражению  $\text{Loves}(\text{John}, \text{Mary})$ .

Остальная часть семантики непосредственно следует из соглашений, принятых нами до сих пор. Поскольку словосочетания  $VP$  представлены как предикаты, желательно придерживаться единобразия и представить глаголы так же, как предикаты. Глагол “*loves*” будет представлен в виде  $\lambda y \lambda x \text{ Loves}(x, y)$ , т.е. в виде предиката, который после подстановки параметра *Mary* возвращает предикат  $\lambda x \text{ Loves}(x, \text{Mary})$ .

В правиле  $VP \rightarrow Verb\ NP$  предикат, представляющий собой семантическую интерпретацию глагола, применяется к объекту, который является семантической интерпретацией словосочетания  $NP$ , для получения семантической интерпретации всего словосочетания  $VP$ . В конечном итоге будет получена грамматика, которая показана в листинге 22.7, и дерево синтаксического анализа, приведенное на рис. 22.7.

**Листинг 22.7.** Грамматика, которая позволяет вывести дерево синтаксического анализа и семантическую интерпретацию для предложения “John loves Mary” (и трех других предложений). Каждая категория дополняется единственным параметром, представляющим семантику

---

$S(\text{rel}(\text{obj})) \rightarrow NP(\text{obj}) \ VP(\text{rel})$   
 $VP(\text{rel}(\text{obj})) \rightarrow \text{Verb}(\text{rel}) \ NP(\text{obj})$   
 $NP(\text{obj}) \rightarrow \text{Name}(\text{obj})$

$\text{Name}(\text{John}) \rightarrow \textbf{John}$   
 $\text{Name}(\text{Mary}) \rightarrow \textbf{Mary}$   
 $\text{Verb}(\lambda x \lambda y \text{ Loves}(x,y)) \rightarrow \textbf{loves}$

---

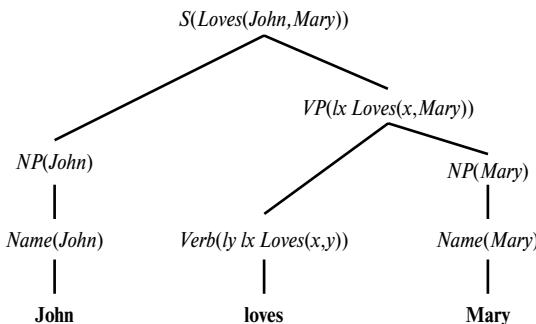


Рис. 22.7. Дерево синтаксического анализа с семантическими интерпретациями для строки “John loves Mary”

### Время события и времена глаголов

Теперь предположим, что необходимо учесть различие между предложениями “John loves Mary” (Джон любит Мэри) и “John loved Mary” (Джон любил Мэри). В английском языке для обозначения относительного времени события используются времена глаголов (прошедшее, настоящее и будущее). Один из удобных вариантов состоит в применении для представления времени событий системы обозначений исчисления событий, описанной в разделе 10.3. В исчислении событий эти два предложения имеют следующие интерпретации:

$$\begin{aligned} e &\in \text{Loves}(\text{John}, \text{Mary}) \wedge \text{During}(\text{Now}, e) \\ e &\in \text{Loves}(\text{John}, \text{Mary}) \wedge \text{After}(\text{Now}, e) \end{aligned}$$

В соответствии с таким подходом два грамматических правила для слов “*loves*” и “*loved*” должны быть такими:

$$\begin{aligned} \text{Verb}(\lambda x \lambda y e \in \text{Loves}(\text{John}, \text{Mary}) \wedge \text{During}(\text{Now}, e)) &\rightarrow \textbf{loves} \\ \text{Verb}(\lambda x \lambda y e \in \text{Loves}(x, y) \wedge \text{After}(\text{Now}, e)) &\rightarrow \textbf{loved} \end{aligned}$$

Не считая этого изменения, все остальные способы представления грамматики остаются теми же самыми, и это — обнадеживающий факт; он говорит о том, что мы на правильном пути, поскольку смогли так легко справиться со сложностями, подобными временам глаголов (хотя фактически мы лишь коснулись поверхности проблемы создания полной грамматики для представления времени событий и врем-

мен глаголов). Используя этот успех в качестве стимулятора, приступим к изучению гораздо более сложной проблемы представления.

## Введение кванторов

Рассмотрим предложение: “Every agent smells a wumpus” (Каждый агент чувствует запах вампуса). Это предложение фактически неоднозначно; основной его смысл состоит в том, что даже если вампусы будут встречаться разным агентам, все они сумеют почувствовать их запах, а еще один вариант смысловой трактовки этого предложения состоит в том, что существует единственный вампус, запах которого чувствует каждый агент<sup>10</sup>. Эти две интерпретации могут быть представлены следующим образом:

$$\begin{aligned} \forall a \ a \in Agents \Rightarrow \\ \exists w \ w \in Wumpuses \wedge \exists e \ e \in Smell(a, w) \wedge During(Now, e) \\ \exists w \ w \in Wumpuses \ \forall a \ a \in Agents \Rightarrow \\ \exists e \ e \in Smell(a, w) \wedge During(Now, e) \end{aligned}$$

На время отложим анализ этой проблемы неоднозначности и рассмотрим только первую интерпретацию. Попытаемся проанализировать ее композиционно, подразделяя все предложение на компоненты *VP* и *NP* следующим образом:

$$\begin{array}{ll} \text{Every agent} & NP(\forall a \ a \in Agents \Rightarrow P) \\ \text{smells a wumpus} & VP(\exists w \ w \in Wumpuses \wedge \\ & \quad \exists e \ (e \in Smell(a, w) \wedge During(Now, e))) \end{array}$$

При этом сразу же возникают две сложности. Первая проблема состоит в том, что семантика всего предложения кажется определяемой семантикой словосочетания *NP*, притом что семантика словосочетания *VP* заполняет только часть *P*. Это означает, что семантику всего предложения невозможно сформировать с помощью конструкции *rel(obj)*. Но это можно сделать с помощью конструкции *obj(rel)*, которая выглядит немного странно (по крайней мере на первый взгляд). Вторая проблема состоит в том, что необходимо получить переменную *a* как параметр отношения *Smell*. Иными словами, семантика предложения формируется путем вставки семантики словосочетания *VP* в правильный слот параметра словосочетания *NP*, и наряду с этим вставки переменной *a* из словосочетания *NP* в правильный слот параметра семантики словосочетания *VP*. Создается такое впечатление, что требуются две функциональные композиции, поэтому вся эта задача обещает стать довольно сложной. Но вся сложность обусловлена тем фактом, что семантическая структура этого предложения значительно отличается от синтаксической структуры.

Для предотвращения подобной путаницы во многих современных грамматиках принят другой подход. В них определена некоторая ~~а~~ промежуточная форма, которая служит в качестве посредника между синтаксисом и семантикой. Эта промежуточная форма имеет два ключевых свойства. Во-первых, она структурно аналогична синтаксису предложения и поэтому может быть легко сконструирована с помощью композиционных средств. Во-вторых, она содержит достаточно информации для того, чтобы ее можно было перевести в обычное высказывание логики первого порядка. Поскольку эта форма занимает положение между синтаксической и логиче-

<sup>10</sup> Если последняя интерпретация кажется маловероятной, рассмотрите следующее утверждение: “Каждый протестант верит только в Бога”.

ской формами, ее называют **квазилогической формой**<sup>11</sup>. В данном разделе будет использоваться квазилогическая форма, которая включает все средства логики первого порядка и дополнена лямбда-выражениями и одной новой конструкцией, которую мы будем называть **квантифицированным термом**. Квантифицированный терм, имеющий семантическую интерпретацию словосочетания “every agent” (каждый агент), записывается следующим образом:

$$[\forall a \ a \in Agents]$$

Эта конструкция выглядит как логическое высказывание, но используется таким же образом, как и логический терм. Интерпретация предложения “Every agent smells a wumpus” в квазилогической форме выглядит следующим образом:

$$\exists e \ (e \in Smell([\forall a \ a \in Agents], [\exists w \ w \in Wumpuses]) \wedge During(Now, e))$$

При создании грамматики, позволяющей производить квазилогическую форму, можно оставить неизменными многие полученные ранее правила. Правило для *S* все еще создает семантику предложения *S* со значением *rel(obj)*. Но некоторые правила действительно изменяются; например, грамматическое правило для артикля “*a*” выглядит таким образом:

$$Article(\exists) \rightarrow a$$

а правило применения артикля в сочетании с существительным является таковым:

$$NP([qx \ sem(x)]) \rightarrow Article(q) \ Noun(sem)$$

Это правило говорит о том, что семантика именного словосочетания *NP* выражается термом с квантором *q*, где квантор обозначен артиклем, включает новую переменную *x* и выражается высказыванием, сформированным путем применения семантики данного имени к переменной *x*. Другие правила для словосочетания *NP* являются аналогичными. В табл. 22.5 приведены семантические типы и примеры форм для каждой синтаксической категории при использовании подхода на основе квазилогической формы. На рис. 22.8 показано дерево синтаксического анализа предложения “every agent smells a wumpus” с использованием этого подхода, а в листинге 22.8 показана полная грамматика.

Таблица 22.5. Таблица, в которой показан тип выражения в квазилогической форме для каждой синтаксической категории. Запись *t* → *x* обозначает функцию, которая принимает параметр типа *t* и возвращает результат типа *x*. Например, семантическим типом для категории *Preposition* является *object*<sup>2</sup> → *sentence*, а это означает, что семантика предлога представляет собой функцию, которая после ее применения к двум логическим объектам позволяет получить логическое высказывание

Категория	Семантический тип	Пример	Квазилогическая форма
<i>S</i>	<i>sentence</i>	I sleep.	$\exists e \ e \in Sleep(Speaker) \wedge During(Now, e)$
<i>NP</i>	<i>object</i>	a dog	$[\exists d \ Dog(d)]$
<i>PP</i>	<i>object</i> <sup>2</sup> → <i>sentence</i>	in [2,2]	$\lambda x \ In(x, [2, 2])$
<i>RelClause</i>	<i>object</i> → <i>sentence</i>	that sees me	$\lambda x \ \exists e \ e \in Sees(x, Speaker) \wedge During(Now, e)$

<sup>11</sup> Некоторые квазилогические формы имеют третье свойство, с помощью которого они приобретают возможность кратко представлять неоднозначности, представимые в логической форме только с помощью длинной дизъюнкции.

Окончание табл. 22.5

Категория	Семантический тип	Пример	Квазилогическая форма
VP	$object^n \rightarrow sentence$	sees me	$\lambda x \exists e e \in Sees(x, Speaker) \wedge During(Now, e)$
Adjective	$object \rightarrow sentence$	smelly	$\lambda x Smelly(x)$
Adverb	$event \rightarrow sentence$	today	$\lambda e During(e, Today)$
Article	quantifier	the	$\exists!$
Conjunction	$sentence^2 \rightarrow sentence$	and	$\lambda p, q (p \wedge q)$
Digit	object	7	7
Noun	$object \rightarrow sentence$	wumpus	$\lambda x x \in Wumpuses$
Preposition	$object^2 \rightarrow sentence$	in	$\lambda x \lambda y In(x, y)$
Pronoun	object	I	$Speaker$
Verb	$object^n \rightarrow sentence$	eats	$\lambda y \lambda x \exists e e \in Eats(x, y) \wedge During(Now, e)$

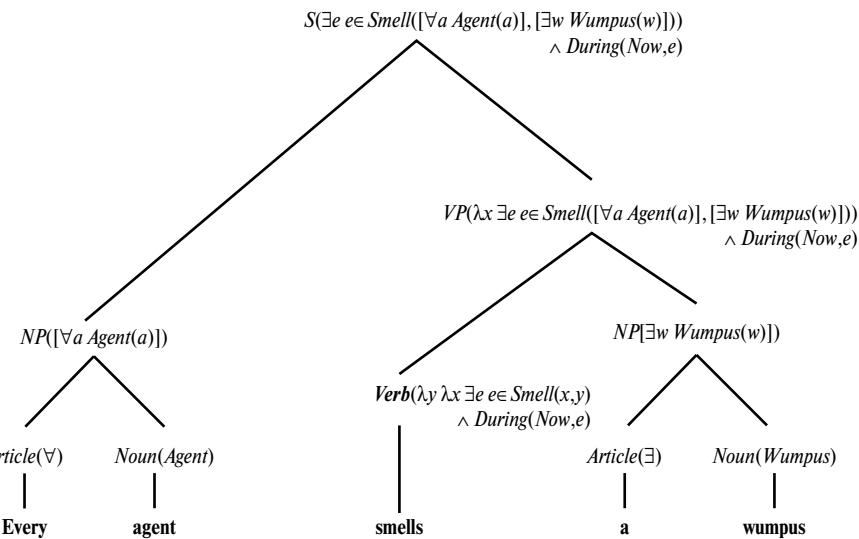


Рис. 22.8. Дерево синтаксического анализа предложения "Every agent smells a wumpus", на котором показана и синтаксическая структура, и семантические интерпретации

### Листинг 22.8. Грамматика с семантикой в квазилогической форме

```

S(rel(obj)) → NP(obj) VP(rel)
S(conj(sem1, sem2)) → S(sem1) Conjunction(conj) S(sem2)
NP(sem) → Pronoun(sem)
NP(sem) → Name(sem)
NP([qx sem(x)]) → Article(q) Noun(sem)
NP([qx obj ∧ rel(x)]) → NP([qx obj]) PP(rel)
NP([qx obj ∧ rel(x)]) → NP([qx obj]) RelClause(rel)
  
```

$$NP([sem_1, sem_2]) \rightarrow Digit(sem_1) \ Digit(sem_2)$$

$$VP(sem) \rightarrow Verb(sem)$$

$$VP(rel(obj)) \rightarrow VP(rel) \ NP(obj)$$

$$VP(sem_1(sem_2)) \rightarrow VP(sem_1) \ Adjective(sem_2)$$

$$VP(sem_1(sem_2)) \rightarrow VP(sem_1) \ PP(sem_2)$$

$$RelClause(sem) \rightarrow \text{that} \ VP(sem)$$

$$PP(\lambda x\{rel(x, obj)\}) \rightarrow Preposition(rel) \ NP(obj)$$

Теперь необходимо преобразовать квазилогическую форму в форму логики первого порядка путем превращения термов с кванторами в настоящие термы. Такая задача может быть решена с помощью простого правила — для каждого терма  $[qx P(x)]$  с квантором  $q$  в квазилогической форме (Quasi-Logical Form — QLF) этот терм с квантором заменить переменной  $x$ , а форму QLF заменить выражением  $qx P(x)$  от QLF, где  $op$  представляет собой операцию  $\Rightarrow$ , если  $q$  — квантор  $\forall$ , и операцию  $\wedge$ , если  $q$  — квантор  $\exists$  или  $\exists!$ . Например, предложение “Every dog has a day” (У каждой собаки бывает праздник) имеет следующую квазилогическую форму:

$$\exists e\{ (e \in Has([\forall d \ d \in Dogs], [\exists a \ a \in Days], Now)) \}$$

В этой форме не указано, какой из двух термов с кванторами должен выйти на передний план, поэтому фактически имеются две возможные логические интерпретации:

$$\forall d \ d \in Dogs \Rightarrow \exists a \ a \in Days \wedge \exists e\{e \in Has(d, a, Now)\}$$

$$\exists a \ a \in Days \wedge \forall d \ d \in Dogs \Rightarrow \exists e\{e \in Has(d, a, Now)\}$$

Первая интерпретация говорит о том, что у каждой собаки бывает свой собственный незабываемый день, а вторая может рассматриваться так, что есть особый день, общий для всех собак. Выбор той или иной интерпретации — это задача этапа устранения неоднозначности. Часто бывает так, что упорядочение термов с кванторами слева направо соответствует упорядочению слева направо самих кванторов, но могут оказывать свое влияние и другие факторы. Преимуществом квазилогической формы является то, что она в сжатом виде представляет все возможности, а ее недостаток состоит в том, что она не помогает сделать выбор между этими возможностями. Для решения такой задачи требуется полная мощь средств устранения неоднозначности, в которых используются все источники свидетельств.

## Прагматическая интерпретация

Выше было показано, как агент может воспринять строку слов и воспользоваться грамматикой для вывода множества возможных семантических интерпретаций. А в этом разделе будет рассматриваться проблема создания полной интерпретации путем добавления контекстно-зависимой информации о текущей ситуации к каждой потенциальной интерпретации.

Наиболее явная потребность в прагматической информации возникает при распознавании смысла **указательных словосочетаний**, представляющих собой словосочетания, которые относятся непосредственно к текущей ситуации. Например,

в предложении “I am in Boston today” (Я сегодня нахожусь в Бостоне) интерпретации указательных словосочетаний “I” и “today” зависят от того, кто и когда произнес это предложение. Мы представляем указательные словосочетания с помощью “констант” (таких как *Speaker*), а фактически они являются **флюентными** словосочетаниями, т.е. зависят от ситуации. Получатель, который воспринимает речевой акт, должен также воспринять информацию о том, кем является отправитель, и использовать эту информацию для определения смысла указательных словосочетаний. Например, получатель может обладать такими знаниями, что  $T(\text{Speaker}=\text{Agent}_B, \text{Now})$ .

Такая команда, как “go to [2,2]”, неявно указывает на получателя. До сих пор в рассматриваемой нами грамматике для  $S$  были определены только повествовательные предложения. Но эту грамматику можно легко дополнить, чтобы она охватывала также команды<sup>12</sup>.

Команда может быть сформирована из словосочетания  $VP$ , в котором в качестве подлежащего неявно указан получатель. Необходимо отличать команды от утверждений, поэтому изменим правила для  $S$ , чтобы включить обозначение типа речевого акта в состав квазилогической формы следующим образом:

$$\begin{aligned} S(\text{Statement}(\text{Speaker}, \text{rel}(\text{obj}))) &\rightarrow NP(\text{obj}) \ VP(\text{rel}) \\ S(\text{Command}(\text{Speaker}, \text{rel}(\text{Hearer}))) &\rightarrow VP(\text{rel}) \end{aligned}$$

Поэтому квазилогическая форма для команды “Go to [2,2]” принимает следующий вид<sup>13</sup>:

$$\text{Command}(\exists e \ e \in Go(\text{Hearer}, [2, 2]))$$

## Применение грамматик DCG для производства языковых конструкций

До сих пор в этой главе изложение было в основном сосредоточено на синтаксическом анализе языка, а не на его производстве. Но тема производства языковых конструкций является не менее интересной. Для выбора правильного фрагмента речи, позволяющего выразить некоторое высказывание, приходится использовать во многом такие же средства, как и при синтаксическом анализе фрагмента речи.

Напомним, что DCG — это система логического программирования, определяющая ограничения, которыми связаны между собой некоторая строка и вариант

<sup>12</sup> Чтобы реализовать полного общающегося агента, требуется также грамматика вопросительных предложений. Но описание вопросительных предложений выходит за рамки настоящей книги, поскольку они налагают **дистанционные зависимости** между составляющими. Например, в предложении “Whom did the agent tell you to give the gold to?” (Кому, согласно сказанному агентом, вы должны отдать золото?) последнее слово “to” должно рассматриваться в ходе синтаксического анализа как словосочетание  $PP$  с отсутствующим словосочетанием  $NP$ ; вместо отсутствующего словосочетания  $NP$  подставлено первое слово предложения, “who”. Для обеспечения того, чтобы отсутствующие словосочетания  $NP$  согласовывались с такими подстановочными словами, применяется сложная система грамматических расширений.

<sup>13</sup> Обратите внимание на то, что квазилогическая форма для команды не включает указание на время события (например, такое как *During(Now, e)*). Это связано с тем, что “go” — фактически форма инфинитива глагола, а не его вариант в настоящем времени. Глагол “go” не позволяет уловить эту разницу, но следует отметить, что правильной формой команды “Ведите себя хорошо!” является “Be good!” (и в ней используется инфинитив глагола “be”), а не “Are good!”. Для обеспечения применения правильных времен глаголов можно дополнить выражения для словосочетаний  $VP$  параметром с обозначением времени и записывать  $VP(\text{rel}, \text{untensed})$  в правой части правила команды, где *untensed* обозначает инфинитив.

синтаксического анализа этой строки. Как известно, определение в формате логического программирования предиката Append может использоваться и для выяснения того, что в выражении  $\text{Append}([1, 2], [3], x)$  задано значение  $x=[1, 2, 3]$ , и для составления списков значений  $x$  и  $y$ , после подстановки которых выражение  $\text{Append}(x, y, [1, 2, 3])$  становится истинным. Аналогичным образом можно записать определение предложения  $S$ , которое может применяться двумя способами — для синтаксического анализа, с целью поиска ответа на запрос  $S(sem, [John, Loves, Mary])$  и получения  $sem=Loves(John, Mary)$ , и для производства синтаксической конструкции путем выдачи запроса  $S(Loves(John, Mary), words)$  и получения  $words=[John, Loves, Mary]$ . Можно также проверить грамматику, введя запрос  $S(sem, words)$  и получив ответ в виде потока пар  $[sem, words]$ , которые производятся с помощью этой грамматики.

Такой подход является вполне применимым в отношении простых грамматик, описанных в этой главе, но может стать более затруднительным при распространении его масштабов на более крупные грамматики. При этом важную роль играет то, какая стратегия поиска используется в машине логического вывода; в частности, применение стратегии поиска в глубину может привести к возникновению бесконечных циклов. Аналогичные предосторожности необходимо принимать, определяя точные детали семантической формы. Может оказаться, что в данной конкретной грамматике не предусмотрен способ выражения логической формы  $X \wedge Y$  для некоторого значения  $X$  и  $Y$ , но есть возможность выразить  $Y \wedge X$ ; такая ситуация свидетельствует о том, что требуется некоторый способ создания канонического представления семантических форм или нужно дополнить процедуру унификации так, чтобы можно было унифицировать выражение  $X \wedge Y$  с выражением  $Y \wedge X$ .

Если в центре внимания находится задача производства синтаксических конструкций, то, как правило, используются более сложные модели производства, которые отличаются от грамматик синтаксического анализа и обеспечивают больший контроль над тем, как именно выражаются компоненты семантики. Одним из подходов, который упрощает задачу переноса центра сосредоточения усилий на наиболее важные части семантической формы, является применение системной грамматики.

## 22.6. НЕОДНОЗНАЧНОСТЬ И УСТРАНЕНИЕ НЕОДНОЗНАЧНОСТИ

В некоторых случаях получатели явно замечают противоречивость некоторого речевого фрагмента. Примеры подобного рода, взятые из заголовков газет, приведены в табл. 22.6.

Но чаще всего воспринимаемый нами язык кажется непротиворечивым. Поэтому, когда исследователи впервые приступили к решению задачи по использованию компьютеров для анализа языка в 1960-х годах, они были весьма удивлены, узнав о том, что ~~почти любой фрагмент речи обладает высокой степенью неоднозначности, даже если альтернативные его интерпретации не являются очевидными для говорящего на родном языке~~. Система, в которой определен большой объем грамматических правил и крупный словарь, может найти тысячи интерпретаций для предложения, которое внешне кажется совершенно обычным. Рассмотрим предложение “The batter hit the ball”, которое на первый взгляд имеет непротиворечивую интерпретацию,

согласно которой игрок в бейсбол ударил по бейсбольному мячу. Но будет получена совсем другая интерпретация, если смысл, который также мог быть выражен в предыдущем предложении, высказан так: “The mad scientist unleashed a tidal wave of cake mix towards the ballroom” (Сумасшедший ученый выпустил в танцевальный зал приливную волну смеси для кекса). Этот пример основан на понятии **лексической неоднозначности**, которое описывает такую ситуацию, что одно слово может иметь больше одного смысла. Лексическая неоднозначность является весьма распространенной; слово “back” может быть наречием (“go back” — иди назад), прилагательным (“back door” — задняя дверь), существительным (“the back of the room” — задняя часть комнаты) или глаголом (“back up your files” — создавай резервные копии своих файлов). Слово “Jack” может представлять собой имя собственное, существительное (игральная карта, шестиконечная металлическая фишка для игры, морской флаг, рыба, самец обезьяны, разъем или устройство для поднятия тяжелых предметов) или глагол (поднимать домкратом автомобиль, охотиться с прожектором или сильно бить по бейсбольному мячу).

**Таблица 22.6. Примеры речевых фрагментов, допускающих противоречивое толкование**

Заголовок англоязычной газеты	Подразумеваемое толкование	Не подразумеваемое толкование
Squad helps dog bite victim	Бригада помогает жертве собачьего укуса	Бригада помогает собаке укусить жертву
Helicopter powered by human flies	В воздухе вертолет, движимый мускульной силой человека	Движителем для вертолета служили человеческие муки
Once-sagging cloth diaper industry saved by full dumps	Пошатнувшиеся котировки акций производителей матерчатых пеленок спасены благодаря появлению полностью впитывающих изделий	Пошатнувшиеся котировки акций производителей матерчатых пеленок спасены благодаря полному демпингу
Portable toilet bombed; police have nothing to go on	Взорван портативный туалет; у полиции нет никаких зацепок	Взорван портативный туалет; полиции нечего предложить взамен
British left waffles on Falkland Islands	Британцы прекратили болтанию о Фолклендских островах	Британцы оставили вафли на Фолклендских островах
Teacher strikes idle kids	Учитель избивает ленивых детей	Из-за забастовок учителей дети начинают лениться
Milk drinkers are turning to powder	Любители молока переходят на порошковое	Любителей молока превращают в порошок
Drunk gets nine months in violin case	В деле со скрипкой пьяного посадили на девять месяцев	Пьяного посадили на девять месяцев в футляр от скрипки

**Синтаксическая неоднозначность** (называемая также **структурной неоднозначностью**) может возникать вместе с лексической неоднозначностью или без нее. Например, строка “I smelled a wumpus in [2,2]” имеет два варианта синтаксического анализа; в одном из них предложное словосочетание “in [2,2]” модифицирует существительное (Я почувствовал запах вампуса, находящегося в квадрате [2,2]), а в другом модифицирует глагол (Находясь в квадрате [2,2], я почувствовал запах вампуса). Синтаксическая неоднозначность приводит к **семантической неоднозначности**, поскольку один вариант синтаксического анализа означает, что в квадрате [2,2] находится вампус, а другой — что в квадрате [2,2] чувствуется его запах. В данном слу-

чае принятие неправильной интерпретации за истинную может оказаться смертельно опасной ошибкой.

Семантическая неоднозначность может обнаруживаться даже в словосочетаниях, не имеющих лексической или синтаксической неоднозначности. Например, именное словосочетание “cat person” может относиться к тому, кто любит представителей семейства кошачьих или является одним из ведущих артистов в фильме “*Attack of the Cat People*”. Словосочетание “coast road” может означать дорогу, которая тянется вдоль морского побережья или ведет к нему.

Наконец, неоднозначность может быть вызвана путаницей между буквальным и переносным значениями. Фигуры речи являются важным элементом поэзии, но на удивление часто встречаются также и в повседневной речи. **Метонимия** — это риторический оборот, в котором один объект используется для обозначения другого. Например, услышав слова: “Chrysler announced a new model” (Компания Крайслер объявила о выпуске новой модели), мы не интерпретируем их как утверждение, что компании могут говорить, поскольку под этими словами подразумевается, что объявление сделано представителем этой компании по связям с общественностью. Метонимия встречается часто и интерпретируется получателями-людьми в основном подсознательно. К сожалению, грамматика, представленная в данной главе, в своем непосредственном виде не позволяет так легко справиться с этой ситуацией. Чтобы учесть семантику метонимии должным образом, необходимо ввести целый новый уровень средств устранения неоднозначности. Для этого требуется предусмотреть два объекта для семантической интерпретации каждого словосочетания в предложении — один для обозначения объекта, на который буквально ссылается это словосочетание (“Chrysler”), а второе — для метонимической ссылки (представитель компании по связям с общественностью). После этого необходимо сформулировать утверждение, что эти два объекта связаны некоторым отношением. В том варианте грамматики, который использовался до сих пор, словосочетание “Chrysler announced” интерпретировалось бы так:

$$\exists x, e \ x = \text{Chrysler} \wedge e \in \text{Announce}(x) \wedge \text{After}(\text{Now}, e)$$

Теперь необходимо преобразовать это высказывание в следующее:

$$\exists m, x, e \ x = \text{Chrysler} \wedge e \in \text{Announce}(m) \wedge \text{After}(\text{Now}, e) \\ \wedge \text{Metonymy}(m, x)$$

В нем говорится о том, что имеется одна сущность  $x$ , равная *Chrysler*, и другая сущность  $m$ , которая объявляет новости от имени компании, и эти две сущности связаны метонимическим отношением. На следующем этапе следует определить, какого рода метонимические отношения могут возникать в предложении. Простейшим случаем является тот, в котором метонимия вообще отсутствует, — буквально указанный объект  $x$  и метонимический объект  $m$  являются идентичными:

$$\forall m, x \ (m = x) \Rightarrow \text{Metonymy}(m, x)$$

В примере с компанией Chrysler приемлемым обобщением является то, что название компании может использоваться вместо имени представителя этой компании по связям с общественностью:

$$\forall m, x \ x \in \text{Organizations} \wedge \text{Spokesperson}(m, x) \Rightarrow \text{Metonymy}(m, x)$$

Другие варианты метонимии могут предусматривать указание имени автора вместо его работ (“I read Shakespeare” — Я читал Шекспира) или, в более общей форме, указание названия компании-производителя вместо ее продукта (“I drive a Honda” — Я вожу Хонду), а также указание части вместо целого (“The Red Sox need a strong arm” — Для Рэд Сокс нужна твердая рука). Некоторые варианты метонимии, такие как “The ham sandwich on Table 4 wants another beer” (Тот бутерброд с ветчиной за четвертым столиком требует еще одно пиво), являются продуктом более поздних изменений в языке и интерпретируются применительно к ситуации.

Правила, описанные в данном разделе, позволяют составить объяснение для предложения “Chrysler announced a new model”, но за самим этим объяснением не следует логический вывод. Для подготовки потенциальных объяснений необходимо использовать средства формирования вероятностных или немонотонных рассуждений.

❖ **Метафора** — это фигура речи, в которой словосочетание с одним буквальным значением используется в качестве указания по аналогии на другое значение. Большинство людей считают, что метафора — это инструмент, используемый поэтами, который не играет большой роли в повседневном общении. Тем не менее целый ряд метафор применяется настолько широко, что мы даже не воспринимаем их как таковые. Одной из таких метафор является представление о том, что “чем больше, тем выше”. Эта метафора позволяет нам говорить о том, что цены растут, повышаются или взмывают вверх, что температура понижается или падает, что чье-то доверие к кому-то рухнуло или что популярность какой-то знаменитости резко подскочила или пошла на убыль.

Существуют два способа, позволяющих учесть наличие подобных метафор. Один из них состоит в том, чтобы включить все знания о метафоре в словарь — добавить новые толкования слов “расти”, “падать”, “повышаться” и т.д., чтобы описать их как относящиеся к количествам любого масштаба, а не только к высоте. Этот подход является приемлемым для многих приложений, но не охватывает обобщающий характер метафоры, которая позволяет людям использовать новые варианты ее реализации, такие как “войти в штопор” или “достичь заоблачных высот”, без опасения быть непонятными. Второй подход состоит в том, что нужно включить в грамматику явно заданные знания об общих метафорах и использовать их для интерпретации новых вариантов применения по мере обнаружения в речи. Например, предположим, что в системе имеется информация о метафоре “чем больше, тем выше”. Иными словами, система может определить, что логические выражения, ссылающиеся на точку шкалы высот, могут интерпретироваться как указывающие на соответствующие точки шкалы количеств. В таком случае выражение “sales are high” (объемы продаж высоки) получит буквальную интерпретацию по типу *Altitude(Sales, High)*, а это выражение можно будет интерпретировать метафорически как *Quantity(Sales, Much)*.

## Устранение неоднозначности

Как было указано выше, ❖ **устранение неоднозначности** — это проблема определения диагноза. Намерение говорящего сообщить информацию становится ненаблюдаемой причиной появления слов в виде фрагмента речи, а задача получателя заключается в том, чтобы проделать эту работу в обратном направлении исходя из сказанных слов и знания ситуации, чтобы определить намерение говорящего с

наибольшей вероятностью. Иными словами, получатель находит решение для следующей задачи:

$$\operatorname{argmax}_{\text{intent}} \text{Likelihood}(\text{intent} \mid \text{words}, \text{situation})$$

где *Likelihood* может представлять собой либо вероятность, либо любую другую числовую меру предпочтения. Предпочтение того или иного рода является необходимым, поскольку правила синтаксической и семантической интерпретации, отдельно взятые, не позволяют выявить уникальную правильную интерпретацию словосочетания или предложения. Поэтому данная работа делится на части: средства синтаксической и семантической интерпретации обеспечивают формирование множества потенциальных интерпретаций, а в процессе устранения неоднозначности среди них выбирается наилучшая.

Обратите внимание на то, что выше было указано на намерение исполнителя речевого акта, а не просто на фактическое высказывание, провозглашаемое говорящим. Например, услышав от политика слова: “I am not a crook”, можно формально назначить вероятность только 50%, что эти слова соответствуют высказыванию о том, что данный политик — не преступник, и 99,999% — высказыванию, что говорящий — не кривой посох пастуха. Тем не менее все равно следует присвоить большую вероятность следующей интерпретации, поскольку скорее всего именно это и подразумевалось:

$$\operatorname{Assert}(\text{Speaker}, \neg(\text{Speaker} \in \text{Criminals}))$$

Рассмотрим еще раз пример неоднозначного предложения: “I smelled a wumpus in [2,2]”. Одной из предпочтительных эвристик является правило ~~правой ассоциации~~, которое гласит, что во время принятия решения о том, где можно поместить в дереве синтаксического анализа словосочетание *PP* “in [2,2]”, следует предпочесть вариант с размещением его в самой правой существующей составляющей, а в данном случае таковой является словосочетание *NP* “a wumpus”. Безусловно, это — всего лишь эвристика; при обработке предложения “I smelled a wumpus with my nose” (Я почувствовал запах вампуса носом) эта эвристика будет перевешиваться тем фактом, что словосочетание *NP* “a wumpus with my nose” (вампус с моим носом) является маловероятным.

Устранение неоднозначности обеспечивается путем комбинирования свидетельств с использованием всех методов представления знаний и формирования рассуждений в условиях неопределенности, которые рассматривались до сих пор в настоящей книге. Все эти знания можно разбить на четыре описанных ниже модели.

- 1. Модель мира.** Вероятность того, что некоторое высказывание является истинным в рассматриваемом мире.
- 2. Мыслительная модель.** Вероятность того, что отправитель оформляет свое намерение сообщить о некотором факте получателю, при условии, что этот факт имел место. В таком подходе комбинируются модели того, в чем уверен отправитель, что думает отправитель о том, в чем уверен получатель, и т.д.
- 3. Языковая модель.** Вероятность того, что будет выбрана определенная строка слов, при том условии, что отправитель имеет намерение сообщить определенный факт. Модели CFG и DCG, представленные в данной главе, имеют

булеву модель правдоподобия; в ней строка может иметь или не иметь определенную интерпретацию. В следующей главе рассматривается вероятностная версия грамматики CFG, позволяющая создать более информационно насыщенную языковую модель для устранения неоднозначности.

4. **Акустическая модель.** Вероятность того, что будет сформирована определенная последовательность звуков с учетом того, что отправитель выбрал данную конкретную строку слов. Задачи распознавания речи рассматриваются в разделе 15.6.

## 22.7. ПОНИМАНИЕ РЕЧИ

❖ **Речь** представляет собой языковой фрагмент произвольной длины, обычно превышающий по длине одно предложение. Как проявления речи рассматриваются учебники, романы, сообщения о погоде и разговоры. До сих пор в этой главе проблемы связной речи в основном игнорировались, поскольку язык проще изучать в лабораторных условиях, разделив речь на отдельные предложения. А в этом разделе предложения рассматриваются в их естественной среде обитания. Он в основном посвящен двум конкретным подзадачам — разрешение ссылок и изучение связной речи.

### Разрешение ссылок

❖ **Разрешение ссылок** представляет собой интерпретацию местоимения или определительного именного словосочетания, которое ссылается на некоторый объект в мире<sup>14</sup>. Разрешение основано на знаниях о мире и на результатах анализа предыдущих фрагментов речи. Рассмотрим следующий отрывок текста:

Джон помахал официанту. Он заказал бутерброд с ветчиной.

Чтобы понять, что слово “он” во втором предложении относится к Джону, необходимо знать, что в первом предложении упоминаются два человека и что Джон играет роль клиента, поэтому заказ, скорее всего, должен сделать он, а не официант. Обычно разрешение ссылок сводится к выбору референта (адресата ссылки) из списка кандидатов, но иногда эта задача требует создания новых кандидатов. Рассмотрим следующий отрывок текста:

После того как Джон сделал предложение Маше, они нашли священника и поженились. Чтобы отпраздновать медовый месяц, они отправились на Гавайи.

Здесь определительное именное словосочетание “медовый месяц” относится к тому, что лишь неявно следует из ситуации, в которой используется глагол “поженились”. Во втором предложении местоимение “они” указывает на группу людей, которая до сих пор еще не была явно обозначена как таковая — Джон и Маша (но не священник).

<sup>14</sup> В лингвистике ссылка на то, что уже было сказано, называется анафорической ссылкой. С другой стороны, ссылка на то, что еще должно быть сказано, называется катафорической ссылкой; таковой является местоимение “он” в предложении “Когда он выиграл свой первый турнир, Тайгеру было 20 лет”.

Выбор наилучшего референта представляет собой процесс устранения неоднозначности, в основе которого лежит использование сочетаний разнообразной синтаксической, семантической и прагматической информации. Некоторые намеки на правильные толкования заданы в форме ограничений. Например, местоимения должны быть согласованы в роде и числе со своими денотатами (словами, которые они заменяют): местоимение “он” может относиться к Джону, но не к Маше; местоимение “они” может относиться к группе, а не кциальному лицу. Местоимения должны также подчиняться синтаксическим ограничениям, определяющим возвратные зависимости. Например, в предложении: “Он увидел его в зеркале” два местоимения должны ссылаться на разных людей, а в предложении: “Он увидел себя” местоимения должны ссылаться на одно и то же лицо. Существуют также ограничения, касающиеся семантической согласованности. Например, в предложении: “Он съел это” местоимение “он” должно ссылаться на то, что ест, а “это” — на то, что едят.

Некоторыми намеками могут служить предпочтения, которые не всегда соблюдаются. Например, если два смежных предложения имеют параллельную структуру, то предпочтительно иметь местоименные ссылки, в которых также соблюдается эта структура. Поэтому в следующем отрывке текста:

Маша прилетела в Сан-Франциско из Нью-Йорка.  
Джон прилетел туда из Бостона.

мы предпочитаем такую трактовку, в которой местоимение “туда” ссылается на Сан-Франциско, поскольку оно играет такую же синтаксическую роль. Если же параллельная структура отсутствует, то подлежащие как денотаты получают предпочтение над дополнениями, поэтому в следующем отрывке текста:

Маша дала Салли указания по дому. Затем она ушла.

предпочтительным денотатом для местоимения “она” является “Маша”, подлежащее первого предложения. Еще один вариант предпочтения распространяется на сущность, которая в процессе речи выступает наиболее ярко. Рассмотрим следующую пару предложений, отдельно взятую:

Дана уронила чашку на тарелку. Она разбилась.

При этом возникает проблема: не ясно, что является референтом местоимения “она” — чашка или тарелка. Но в более широком контексте эта неоднозначность устраняется:

Дана очень любила синюю чашку. Эта чашка была подарена близким другом. К сожалению, однажды, накрывая на стол, Дану уронила чашку на тарелку. Она разбилась.

В данном случае фокусом внимания является чашка, и поэтому рассматривается как предпочтительный референт.

Был разработан целый ряд алгоритмов разрешения ссылок. Особенно замечательным является один из первых таких алгоритмов, разработанный Хоббсом [662], поскольку он позволил достичь необычно высокой в то время степени статистической достоверности. Хоббс использовал тексты трех разных жанров и сообщил о том, что с помощью этого алгоритма достигнута точность 92%. Условием проведения этих экспериментов должны были стать правильные результаты синтаксического анализа, выработанные синтаксическим анализатором; не имея такового в своем распоряжении, Хоббс составлял деревья синтаксического анализа вручную.

Алгоритм Хоббса действует на основе поиска: в нем осуществляется поиск в предложениях, начиная от текущего предложения и двигаясь в обратном направлении. Такой метод гарантирует, что в первую очередь будут рассматриваться последние по времени определенные кандидаты. Поиск в предложении осуществляется в ширину, слева направо. Это гарантирует, что подлежащие будут рассматриваться перед дополнениями. В алгоритме выбирается самый первый кандидат, который удовлетворяет описанным в этом абзаце ограничениям.

### Структура связной речи

Откройте данную книгу на 10 случайно выбранных страницах и запишите на бумагу первое полное предложение с каждой страницы. Полученный текст обязательно будет несвязным. Аналогичным образом, если вы возьмете связный отрывок текста, состоящий из 10 предложений, после чего переставите эти предложения случайным образом, то получите несвязный текст. Такой эксперимент показывает, что предложения в речи на естественном языке весьма отличаются от высказываний в логике. В логике после ввода в базу знаний с помощью предиката `Tell` высказываний *A*, *B* и *C* в любом порядке будет получена конъюнкция *A*  $\wedge$  *B*  $\wedge$  *C*. *A* в естественном языке играет роль порядок предложений; в качестве примера можно указать, насколько отличаются следующие два ответа на просьбу показать дорогу: “Пройдите два квартала. Поверните направо” и “Поверните направо. Пройдите два квартала”.

Речь обладает структурой, возвышающейся над уровнем предложения. Эту структуру можно исследовать с помощью грамматики речи:

$$\text{Segment}(x) \rightarrow S(x)$$

$$\text{Segment}(\text{CoherenceRelation}(x, y)) \rightarrow \text{Segment}(x) \text{ Segment}(y)$$

Приведенные выше грамматические правила указывают, что речь состоит из отрывков, где каждый отрывок представляет собой либо предложение, либо группу предложений, а сами отрывки связаны **отношениями связности**. В отрывке текста “Пройдите два квартала. Поверните направо” отношение связности состоит в том, что действие, указанное в первом предложении, создает предпосылки действия во втором предложении: получатель должен повернуть направо, только пройдя два квартала. Разные исследователи предложили различные варианты определения отношений связности; один из типичных перечней таких отношений приведен в табл. 22.7. Теперь рассмотрим приведенный ниже рассказ.

1. Вчера произошло забавное событие.
2. Джон отправился в модный ресторан.
3. Он заказал утку.
4. Счет составил почти 50 долларов.
5. Джон буквально испытал шок, обнаружив, что у него нет денег.
6. Он забыл кошелек дома.
7. Официант сказал, что можно заплатить позже.
8. Но он очень расстроился из-за своей забывчивости.

**Таблица 22.7. Перечень отношений связности, взятый из [663]. Каждое отношение связывает два смежных фрагмента текста,  $S_1$  и  $S_2$**

Отношение связности	Описание	Пример	Примечание
Описание предпосылки или причины	Фрагмент текста $S_1$ отражает изменение состояния (которое может быть неявным), ставшее предпосылкой или причиной того, что описано во фрагменте текста $S_2$	“I went outside. I drove to school” (Я вышел из дома. Я поехал в школу.)	Выход из дома неявно создает предпосылки для поездки в автомобиле
Объяснение — отношение связности, обратное описанию предпосылки	Фрагмент текста $S_2$ описывает предпосылку или причину того, чем вызвана ситуация, представленная во фрагменте текста $S_1$ , и тем самым служит для нее объяснением	“I was late for school. I overslept” (Я опоздал в школу. Я проспал.)	
Подготовительное описание	Фрагмент текста $S_1$ описывает обстановку или предысторию для фрагмента текста $S_2$	“It was a dark and stormy night. Rest of story” (Была темная и ветреная ночь. <i>Остальная часть рассказа.</i> )	
Оценка	Из фрагмента текста $S_2$ следует, что фрагмент текста $S_1$ является реализацией плана говорящего по его использованию в составе речевого акта	“A funny thing happened. Rest of story” (Однажды произошло забавное событие. <i>Остальная часть рассказа.</i> )	
Применение в качестве примера	Фрагмент текста $S_2$ представляет собой пример общего принципа, описанного во фрагменте текста $S_1$	“This algorithm reverses a list. The input $[A, B, C]$ is mapped to $[C, B, A]$ ” (Этот алгоритм служит для обращения списка. Входные данные $[A, B, C]$ преобразуются в $[C, B, A]$ .)	
Обобщение	Фрагмент текста $S_1$ представляет собой пример общего принципа, описанного во фрагменте текста $S_2$	“ $[A, B, C]$ is mapped to $[C, B, A]$ . In general, the algorithm reverses a list” (Входные данные $[A, B, C]$ преобразуются в $[C, B, A]$ . В общем этот алгоритм служит для обращения списка.)	
Констатация того факта, что ожидания не оправдались	Вывод на основании фрагмента текста $S_2$ следствия $\neg P$ , отрицающего следствие $P$ , которое при обычных обстоятельствах должно было быть получено на основании фрагмента текста $S_1$	“This paper is weak. On the other hand, it is interesting” (Эта статья слаба. С другой стороны, она представляет интерес.)	

Здесь предложение 1 находится в положении отношения оценки для остальной части речи; предложение 1 представляет собой метакомментарий говорящего ко всему рассказу. Предложение 2 создает предпосылки предложения 3, а предложения 2 и 3, вместе взятые, указывают причину для предложения 4 с учетом того неявного промежуточного состояния, что Джон съел утку. Теперь предложения 2–4 становятся обоснованием остальной части рассказа. Предложение 6 служит объяснением предложения 5, в предложения 5 и 6 являются предпосылкой для 7. Следует подчеркнуть, что они являются предпосылкой, а не причиной, поскольку офицант мог среагировать на эту ситуацию иначе. Предложения 5–7, вместе взятые, становятся причиной для предложения 8. В упр. 22.13 предлагается нарисовать дерево синтаксического анализа для этого рассказа.

Отношения связности позволяют связать речь в единое целое. Они служат для говорящего руководящими указаниями при принятии решения о том, что следует сказать и что оставить не выраженным явно, а также служат руководством для получателя при восстановлении информации о намерениях говорящего. Отношения связности могут служить в качестве фильтра при устранении неоднозначности предложений: предложения, отдельно взятые, могут оказаться неоднозначными, но большинство из этих неоднозначных интерпретаций перестают согласовываться друг с другом в связной речи.

До сих пор вопросы разрешения ссылок и структуры речи рассматривались раздельно. Но фактически эти два вопроса тесно переплетаются. Например, в теории Гроша и Сиднера [598] учитывается, на что направлено внимание говорящего и слушающего во время речи. В их теорию включено описание стека  $\bowtie$  пространств фокусов. Некоторые фрагменты речи вызывают сдвиг фокуса путем заталкивания или выталкивания элементов из стека. Например, в истории с рестораном предложение: “Джон отправился в модный ресторан” заталкивает в стек новый фокус внимания. В пределах данного фокуса отправитель может использовать определенное словосочетание *NP*, чтобы указать на “этого официанта” (а не просто на “какого-то официанта”). Если бы рассказ был продолжен словами: “Джон отправился домой”, то данное пространство фокусов было бы вытолкнуто из стека и в рассказе больше нельзя было бы указывать на официанта словами “этот официант” или “он”.

## 22.8. ИНДУКТИВНЫЙ ВЫВОД ГРАММАТИКИ

$\bowtie$  **Индуктивный вывод грамматики** представляет собой задачу определения с помощью обучения грамматики на основе данных. Очевидно, что попытка решить такую задачу вполне оправдана, поскольку практика показала, насколько сложно конструировать грамматику вручную, а в Internet можно бесплатно получить миллиарды готовых примеров фрагментов речи. Но такая задача является сложной, поскольку пространство возможных грамматик бесконечно, а проверка того, что данная конкретная грамматика производит необходимое множество предложений, требует больших вычислительных затрат.

Одной из интересных моделей является система Sequitur [1122]. Для нее не требуется никаких входных данных, кроме одного отрывка текста (который не требуется даже предварительно разбивать на предложения). Эта система формирует грамматику в очень специализированной форме — грамматику, которая производит лишь

единственную строку, а именно первоначальный текст. Иначе к анализу результатов этой системы можно подойти таким образом, что Sequitur в процессе обучения определяет лишь такой объем грамматических правил, который является достаточным для синтаксического анализа данного текста. Ниже приведена приемлемая разбивка с помощью скобок, обнаруженная этой системой для одного предложения из более крупного текста с информацией о новостях.

```
[Most Labour] [sentiment [[would still] [favor the] abolition]] [[of  
[the House]] [of Lords]]
```

В этом фрагменте правильно выделены такие составляющие, как словосочетание *PP* “of the House of Lords”, хотя и допущены отклонения от традиционного метода анализа, например artikel “the” включен в одну группу с предшествующим глаголом, а не со следующим существительным.

Система Sequitur основана на той идее, что “хорошая грамматика — компактная грамматика”. В частности, в ней принудительно применяются следующие два ограничения. Во-первых, ни одна пара смежных символов не должна появляться в грамматике больше одного раза. Если пара символов *A B* появляется в правой части нескольких правил, то эту пару следует заменить новым нетерминальным символом, называть его, допустим, *C* и ввести правило  $C \rightarrow A B$ . Во-вторых, каждое правило должно использоваться по меньшей мере дважды. Если некоторый нетерминальный символ *C* появляется в грамматике только один раз, то необходимо удалить правило для *C* и заменить его единственное вхождение правой частью данного правила. Эти два ограничения применяются в процедуре жадного поиска, которая сканирует текст слева направо, инкрементно наращивая в ходе этого грамматику и налагая ограничения, как только появляется такая возможность. В табл. 22.8 показано, как действует этот алгоритм применительно к входному тексту “*abcdabcabc*”. Алгоритм восстанавливает оптимально компактную грамматику для этого текста.

Таблица 22.8. Трассировка выполнения задания, в котором система Sequitur осуществляет индуктивный вывод грамматики для входного текста “*abcdabcabc*”. Работа начинается с введения правила для *S* и добавления каждого символа к концу этого правила по очереди. После добавления шестого символа обнаруживается первое вхождение повторяющейся пары: *bc*. Поэтому оба вхождения *bc* заменяются новым нетерминальным символом *A* и вводится правило  $A \rightarrow bc$ . После добавления еще трех символов девятый символ вызывает еще одно повторение *bc*, поэтому такая пара снова заменяется символом *A*. Это приводит к появлению двух вхождений символов *aA*, поэтому они заменяются новым нетерминальным символом *B*. После добавления десятого и последнего терминального символа обнаруживаются два вхождения *Bd*, поэтому они заменяются новым нетерминальным символом *C*. Но теперь символ *B* появляется только один раз, в правой части правила *C*, поэтому *B* заменяется его расширением *aA*.

Входные данные	Грамматика	Комментарии
1 <i>a</i>	$S \rightarrow a$	
2 <i>ab</i>	$S \rightarrow ab$	
3 <i>abc</i>	$S \rightarrow abc$	
4 <i>abcd</i>	$S \rightarrow abcd$	
5 <i>abcdb</i>	$S \rightarrow abcdb$	
6 <i>abcdbc</i>	$S \rightarrow abcdcbc$	<i>bc</i> встречается дважды
	$S \rightarrow aAdA; A \rightarrow bc$	

Окончание табл. 22.8

Входные данные	Грамматика	Комментарии
7 <i>abcdcba</i>	$S \rightarrow aAdAa; A \rightarrow bc$	
8 <i>abcbcab</i>	$S \rightarrow aAdAab; A \rightarrow bc$	
9 <i>abcbcababc</i>	$S \rightarrow aAdAabc; A \rightarrow bc$ $S \rightarrow aAdAaA; A \rightarrow bc$ $S \rightarrow BdAB; A \rightarrow bc; B \rightarrow aA$	<i>bc</i> встречается дважды <i>aA</i> встречается дважды
10 <i>abcbcabcd</i>	$S \rightarrow BdABd; A \rightarrow bc; B \rightarrow aA;$ $S \rightarrow CAC; A \rightarrow bc; B \rightarrow aA;$ $C \rightarrow Bd$ $S \rightarrow CAC; A \rightarrow bc; C \rightarrow aAd$	<i>Bd</i> встречается дважды <i>B</i> встречается только один раз

В следующей главе будут описаны другие алгоритмы индуктивного вывода грамматики, которые могут применяться к вероятностным контекстно-свободным грамматикам. А теперь обратимся к проблеме определения с помощью обучения грамматики, которая дополнена семантикой. Поскольку расширенная грамматика представляет собой логическую программу, состоящую из хорновских выражений, то для этого могут применяться методы индуктивного логического программирования. Одной из программ индуктивного логического программирования (Inductive Logic Programming — ILP) является Chill [1640], которая на основании примеров определяет с помощью обучения грамматику и специализированный синтаксический анализатор для этой грамматики. Целевой проблемной областью являются запросы к базе данных на естественном языке. Обучающие примеры состоят из пар, в которые собраны строки из слов и соответствующих запросов, например, как показано ниже.

% Как называется столица штата с наибольшим количеством населения?  
 What is the capital of the state with the largest population?  
 Answer(*c*, Capital(*s*, *c*)  $\wedge$  Largest(*p*, State(*s*)  $\wedge$  Population(*s*, *p*)))

Задача программы Chill состоит в том, чтобы определить с помощью обучения предикат Parse(*words*, *query*), который является совместимым с примерами и, можно надеяться, допускает приемлемое обобщение применительно к другим примерам. Непосредственное использование метода ILP для определения этого предиката с помощью обучения приводит к достижению слишком низкой производительности: полученный с помощью индуктивного вывода синтаксический анализатор имеет точность лишь приблизительно 20%. Но, к счастью, системы обучения с помощью ILP могут совершенствоваться путем введения дополнительных знаний. В данном случае большая часть предиката Parse была определена в виде логической программы и задача системы Chill свелась к тому, что нужно было осуществить логический вывод правил управления, которыми может руководствоваться синтаксический анализатор при выборе одного варианта синтаксического анализа перед другим. После введения этих дополнительных знаний система Chill достигла точности от 70 до 85% при решении различных задач обработки запросов к базе данных.

## 22.9. РЕЗЮМЕ

Разработка способов понимания естественного языка является одним из наиболее важных направлений развития искусственного интеллекта. Это направление подпитывается идеями, взятыми из философии и лингвистики, а также обогащается методами логического и вероятностного представления знаний и формирования рассуждений. В отличие от других областей искусственного интеллекта, для понимания естественного языка требуется эмпирическое исследование фактического поведения людей, что, в свою очередь, представляет собой сложную и интересную задачу.

- Агенты посылают друг другу сигналы, чтобы добиться определенных целей: проинформировать, предупредить, попросить помочь, поделиться знаниями или что-то пообещать. Отправка сигнала таким образом называется **речевым актом**. В конечном итоге все речевые акты представляют собой попытку заставить других агентов во что-то поверить или что-то сделать.
- Язык состоит из принятых в соответствии с соглашениями **знаков**, которые несут в себе смысл. Многие животные используют знаки только в их непосредственном виде. А люди, по-видимому, являются единственной разновидностью животных, которая применяет **грамматику** для производства структурированных сообщений, характеризующихся неограниченным разнообразием.
- Общение требует от говорящего выполнения трех этапов: достижение намерения донести некоторую идею, осуществляемое в уме производство цепочки слов и их физический синтез. После этого слушающий должен выполнить четыре этапа: восприятие, синтаксический анализ, устранение неоднозначности и усвоение смысла. Все варианты использования языка являются **ситуационными**; под этим подразумевается, что смысл фрагмента речи может зависеть от той ситуации, в которой он был произведен.
- Полезными инструментальными средствами, позволяющими учитывать некоторые аспекты естественного языка, являются формальная теория языка и грамматики **структуры словосочетаний** (в частности, **контекстно-свободные грамматики**).
- Синтаксический анализ предложений, выраженных на контекстно-свободном языке, может осуществляться за время  $O(n^3)$  с помощью **диаграммного синтаксического анализатора**.
- Чтобы можно было проще справиться с задачами согласования подлежащего и глагольного сказуемого, а также выбора падежа местоимения, удобно воспользоваться таким методом, как **расширение** грамматики. Применение необходимых расширений обеспечивается с помощью формальной системы, представляющей собой **грамматику определенных выражений** (Definite Clause Grammar — DCG). Грамматика DCG обеспечивает синтаксический анализ и семантическую интерпретацию (и даже производство) текста с помощью логического вывода.
- С помощью расширенной грамматики может также осуществляться **семантическая интерпретация**. Одним из удобных посредников между деревьями синтаксического анализа и семантическими представлениями может стать квази-логическая форма.

- Очень важной проблемой при понимании естественного языка является **неоднозначность**; большинство предложений имеют много возможных интерпретаций, но обычно подходящей является только одна из них. Устранение неоднозначности основано на знаниях о мире, о текущей ситуации и о нормативном использовании языка.
- Большинство языков существуют в контексте множества предложений, а не только одного предложения. В основе исследования связных текстов лежит понятие **речи**. В этой главе описано, как можно разрешать местоименные ссылки, охватывающие несколько предложений, и показано, благодаря чему предложения соединяются в связные отрывки текста.
- Метод **индуктивного вывода грамматики** позволяет определять с помощью обучения грамматику на основании примеров, хотя и существуют ограничения, касающиеся того, насколько успешно может быть обобщена эта грамматика.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Наука о знаках и символах как элементах языка была названа **семиотикой** Джоном Локком [941], хотя она не получила развития до XX столетия [353], [1198]. К новейшим обзорным исследованиям относятся [273] и [429].

Идея языка как действия была сформулирована в рамках философских исследований лингвистической направленности, проводимых в XX веке [49], [596], [1608], и особенно ярко отразилась в книге *Speech Acts* [1375]. Праородителем идеи речевых актов был Протагор (ок. 430 г. до н.э.), который определил четыре типа предложений: просьба, вопрос, ответ и приказ. Модель речевого акта, основанная на плане, была впервые предложена в [278]. Связь между языком и действием исследовалась с использованием распознавания плана для понимания рассказов Виленским [1591]. В [277] собраны более современные работы в этой области.

Как и семантические сети, контекстно-свободные грамматики (называемые также *грамматиками структуры словосочетаний*) представляют собой повторное изобретение метода, который был впервые использован древнеиндийскими филологами (особенно Панини, ок. 350 г. до н.э.), изучающими шастрический санскрит [716]. Они были повторно изобретены Ноамом Хомским [250] для анализа английского синтаксиса и независимо от него Джоном Бэкусом для анализа синтаксиса языка Algol-58. Наур [1116] расширил систему обозначений Бэкуса, и теперь его заслуги отмечены тем, что буква “N” в обозначении формы BNF, которое первоначально расшифровывалось как “Backus Normal Form” (нормальная форма Бэкуса), считается сокращением от его фамилии [58]. Одну из разновидностей расширенной грамматики, называемой **семиотикой атрибутов**, которая удобна для представления языков программирования, предложил Кнут [808]. Грамматики определенных выражений были введены в научный обиход Колмерором [284], а в дальнейшем доработаны и популяризированы Перейрой и Уорреном [1208]. Язык программирования Prolog был разработан Аленом Колмерором специально для решения задачи синтаксического анализа французского языка. Колмерор фактически ввел в действие формальную систему, называемую *грамматикой метаморфоз*, которая превосходила по

своим возможностям грамматику определенных выражений, но вскоре после этого появилась более практическая грамматика DCG.

Было предпринято много попыток написания формальных грамматик естественных языков, как в “чистой” лингвистике, так и в вычислительной лингвистике. К машинно-ориентированным грамматикам такого типа относятся системы, разработанные в рамках проекта Linguistic String Project в Университете штата Нью-Йорк [1343] и проекта XTAG в Университете штата Пенсильвания [403]. Хорошим примером современной системы DCG может служить Core Language Engine [22]. Существует также несколько исчерпывающих, но неформальных грамматик английского языка [701], [735], [1015], [1261]. К хорошим учебникам по лингвистике относятся введение в синтаксис [1342] и учебники по семантике [249], [643]; [1016] в основном посвящена описанию логики и рассчитана на лингвистов.

С середины 1980-х годов наметилась тенденция к тому, что больше информации стали вводить в лексикон и меньше в грамматику. Первой крупной грамматической формальной системой, которая характеризовалась высокой степенью лексикализации, была лексически-функциональная грамматика, или сокращенно LFG (Lexical-Functional Grammar) [183]. Доведение процесса лексикализации до предела приводит к созданию **категориальной грамматики**, в которой количество грамматических правил может стать крайне малым, например равным двум, или **грамматики зависимостей** [1033], в которой не существует словосочетаний, а есть только слова. В [1431] описан широко применяемый синтаксический анализатор, в котором используется грамматика зависимостей. Грамматика соединения деревьев, или сокращенно TAG (Tree-Adjoining Grammar) [749], строго говоря, не является лексической, но получила широкое распространение в своей лексикализованной форме [1356]. Интерес представляет общедоступный словарь Wordnet [462], состоящий примерно из 100 000 слов и словосочетаний, классифицированных по частям речи и связанных с помощью семантических отношений, таких как “синоним”, “антоним” и “часть–целое”.

Первые компьютеризированные алгоритмы синтаксического анализа были продемонстрированы в [1632]. Эффективные алгоритмы были разработаны в конце 1960-х годов, и с тех пор в них было введено лишь немного дополнений [587], [773], [1636]. Рассматриваемый в настоящей главе диаграммный синтаксический анализатор в большей степени соответствует описанному в [427]. Хороший общий обзор по этой теме приведен в книге Ахо и Ульмана [9], посвященной синтаксическому анализу и компиляции. В [1001] показано, как в обычной ситуации можно добиться высокой эффективности алгоритма диаграммного синтаксического анализа с дополнениями. В [256] рассматривается проблема устранения синтаксической неоднозначности.

Направление исследований по формальной семантической интерпретации естественных языков впервые возникло в рамках философии и формальной логики и особенно тесно связано с работой Альфреда Тарского [1490] по семантике формальных языков. Бар-Хиллел впервые проанализировал проблемы прагматики и высказал предположение, что они могут быть решены с помощью формальной логики. В частности, он ввел в лингвистику предложенный Ч. С. Пирсом [1198] термин “индексальный” (indexical), т.е. обладающий смыслом только в непосредственном контексте своего применения [68]. Очерк Ричарда Монтея *English as a formal language* (Английский как формальный язык) [1071] представляет собой своего рода манифест сторонников логического анализа языка, но более доступными для восприятия являются [408] и [924]. Полный сборник трудов Монтея вышел под редакцией Томасона

[1505]. В искусственном интеллекте традиции Монтегю продолжили Макаллестер и Гивен [1007], которые разработали много новых формальных методов.

Идея использования промежуточной, или квазиологической формы, для решения таких проблем, как определение области действия кванторов, впервые выдвинута Вудсом [1614] и в настоящее время применяется во многих современных системах [22], [714].

Первой системой NLP, предназначеннной для решения реальной задачи, по-видимому, стала система формирования ответов на вопросы Baseball [590], которая выдавала ответы на вопросы, касающиеся базы данных со статистическими сведениями о бейсболе. Вскоре после этого была разработана система Вудса Lunar [1612], которая отвечала на вопросы об образцах лунного грунта, доставленного на Землю в рамках программы Аполлон. Роджер Шенк со своими студентами создал ряд программ [425], [1358], [1359], [1590], предназначенных для решения задачи понимания языка. Но при разработке этих программ основное внимание было сосредоточено не на языке как таковом, а, скорее, на представлении знаний и формировании рассуждений. К числу рассматриваемых проблем относилось представление стереотипных ситуаций [314], описание организации человеческой памяти [829], [1287], а также понимание планов и целей [1591].

Задачи производства текстов на естественном языке рассматривались с самых первых дней развития работ по машинному переводу, начиная с 1950-го года, но не были сформулированы в связи с потребностями выработки текста на одном языке (а не двух, как при переводе) до 1970-х годов. Характерными исследованиями в этом направлении являются работы, описанные в [571] и [1413]. Одной из первых полномасштабных систем производства текста явилась система Penman [80], основанная на системной грамматике [775]. В 1990-х годах появились две важные общедоступные системы выработки текста, KPML [79] и FUF [433]. К числу наиболее важных книг по производству текста относятся [690], [1029], [1183] и [1275].

Одной из самых ранних работ по устранению неоднозначности явилось исследование Уилкса [1596] по теории **семантики предпочтений**, в котором предпринимались попытки поиска интерпретаций, позволяющих свести к минимуму количество семантических аномалий. В [661] описана система аналогичного назначения, которая ближе к композиционной семантике, описанной в этой главе. В [666] представлена количественная инфраструктура для измерения качества синтаксической и семантической интерпретации. С тех пор получили более широкое распространение методы, основанные на использовании явной байесовской инфраструктуры [238], [1625]. В лингвистике получила распространение теория оптимальности (Linguistics) [761], основанная на идее формирования мягких ограничений, налагаемых на грамматику, что позволяет выполнять естественное ранжирование интерпретаций, а не использовать грамматику для производства всех возможных вариантов с равным рангом. В [1147] рассматриваются проблемы изучения многочисленных одновременных интерпретаций как метода, применяемого вместо выбора одной интерпретации с максимальным правдоподобием. Литературные критики [437], [663] выразили сомнение в том, удастся ли когда-либо решить проблему устранения или уменьшения неоднозначности.

Формальная модель метонимии представлена в [1150]. В [883] приведены соответствующие результаты анализа и описан каталог метафор, широко применяемых в английском языке. В [1160] представлен сборник статей по метафорам, а в [992] предложен вычислительный подход к интерпретации метафор.

Приведенная в этой главе трактовка разрешения ссылок основана на работе Хоббса [662]. Более сложная модель, предложенная в [888], основана на механизме присваивания количественных оценок. В опубликованных немного позже работах [529], [789] использовалось машинное обучение для настройки количественных параметров. Двумя превосходными обзорами проблематики разрешения ссылок являются книги [660] и [1067].

В изданной в 1758 году книге Дэвида Юма *Enquiry Concerning the Human Understanding* утверждалось, что речь становится связной благодаря действию “трех принципов связи между идеями, а именно: сходство, смежность во времени или пространстве, а также причина или результат”. С этого началась долгая история попыток определить отношения связности речи. Множество отношений, которое используется в данной главе, предложено Хоббсом [663]; в [974] представлено более широкое множество, которое включает готовность к принятию решения, свидетельство, обоснование, мотивацию, основание, следствие, предоставление возможностей, использование возможностей, переформулировку утверждения, условие, обстоятельство, причину, соглашение, предысторию и тезис–антитезис. Развитие этой модели привело к созданию теории риторической структуры (Rhetorical Structure Theory — RST), которая, по-видимому, является одной из наиболее перспективных современных теорий [975]. Некоторые примеры, приведенные в настоящей главе, заимствованы из книги, принадлежащей перу Эндрю Келера [756].

В [598] представлена теория связности речи, основанная на изучении сдвига фокуса внимания слушателя, а в [597] предложена близкая к ней теория, основанная на понятии сосредоточения внимания. В [750] собраны важные ранние работы по проблемам речи. Веббер представил модель взаимодействия ограничений синтаксиса и речи в том, что может быть высказано в любом моменте речи [1560], а также описал способ взаимодействия с содержимым речи времен глаголов [1561].

Первый важный результат по **индуктивному выводу грамматики** оказался отрицательным: Голд [569] показал, что нельзя надежно определить с помощью обучения правильный вариант контекстно-свободной грамматики по множеству строк, полученных с помощью этой грамматики. По сути, его идея состоит в том, что если дано множество строк  $s_1, s_2, \dots, s_n$ , то правильная грамматика может либо оказаться всеобъемлющей ( $S \rightarrow word^*$ ), либо стать копией входных данных ( $S \rightarrow s_1 \mid s_2 \mid \dots \mid s_n$ ), либо занять какое-то промежуточное положение. Выдающие лингвисты, такие как Хомский [251], [252] и Пинкер [1211], [1213], использовали результат Голда для доказательства того, что должна существовать врожденная **универсальная грамматика**, которой все дети владеют от рождения. При этом особый интерес представляет так называемый *довод о бедности стимула* (Poverty of the Stimulus), согласно которому дети не получают другой входной языковой информации, кроме положительных примеров: их родители и сверстники главным образом вырабатывают точные примеры используемого ими языка и очень редко исправляют ошибки. Таким образом, поскольку Голд доказал, что определение с помощью обучения контекстно-свободной грамматики на основании положительных примеров является невозможным, то дети уже обязаны “знать” эту грамматику и в процессе обучения языку просто настраивают некоторые параметры этой врожденной грамматики и учат словарь. Хотя эти доводы продолжают повторять многие лингвисты школы Хомского, они были опровергнуты некоторыми другими лингвистами [436], [1243] и большинством ученых, работающих в области компьютерных наук. Еще в 1969 году Хорнинг [679] показал, что с помощью такого

метода, как РАС-обучение, можно определить с помощью обучения вероятностную контекстно-свободную грамматику. С тех пор было проведено много убедительных эмпирических демонстраций успешного обучения на основании только положительных примеров, таких как работы в области ILP [1075] и [1101], а также замечательные докторские диссертации [350] и [1370]. С помощью обучения возможно также определить и другие грамматические формальные системы, такие как регулярные языки [386], [1157], регулярные древовидные языки [226] и конечные автоматы [1173].

Система Sequitur разработана Невилл-Маннингом и Уиттеном [1122]. Интересно отметить, что эти авторы, так же как и де Маркен, указали, что предложенные ими схемы индуктивного вывода грамматики представляют собой одновременно хорошие схемы сжатия. Этот результат соответствует принципу кодирования с минимальной длиной описания: из определения качественной грамматики следует, что она должна минимизировать сумму двух значений длины: длины грамматики и длины дерева синтаксического анализа текста.

К работам в области индуктивного логического программирования, относящимся к определению языка с помощью обучения, принадлежат система Chill [1640] и программа Муни и Калиффа [1076]; эти работы позволили определить правила для прошедшего времени глаголов лучше, чем до сих пор удавалось добиться с помощью нейронных сетей или систем деревьев решений. [315] представляет собой отредактированный сборник статей по определению с помощью обучения языка средствами логики.

Ассоциация ACL (Association for Computational Linguistics) проводит регулярные конференции и публикует журнал *Computational Linguistics*. Проводится также конференция *International Conference on Computational Linguistics* (COLING). Многие важные ранние статьи собраны в антологии *Readings in Natural Language Processing* [599]. В [321] основное внимание удалено описанию инstrumentальных средств, практически применимых для создания систем NLP. В [756] приведено исчерпывающее введение в эту область, [18] посвящена описанию работ, проведенных немного раньше. В [298] и [1207] приведен краткий обзор по синтаксической обработке с помощью различных реализаций на языке Prolog. Много полезных статей с описанием этой области приведено в книге *Encyclopedia of AI*; особого внимания заслуживают статьи “*Computational Linguistics*” и “*Natural Language Understanding*”.

## УПРАЖНЕНИЯ

- 22.1.** Прочитайте следующий текст с описанием процедуры, состоящей из четырех этапов, чтобы понять его смысл, и запомните столько, сколько сможете. Достигнутые вами успехи будут оценены позже.

Эта процедура фактически весьма проста. Сначала нужно распределить вещи по разным группам. Конечно, для одного раза может оказаться достаточно одной стопки, в зависимости от того, сколько еще предстоит сделать. Если вам потребуется отправиться куда-то еще из-за нехватки необходимых средств, то в этом будет состоять следующий этап, а если нет, то вы уже достаточно хорошо подготовились. Набирая вещи, важно не переусердствовать. Это означает, что лучше сразу проделывать эту процедуру со слишком малым, чем со слишком большим

количеством вещей. Если прошло еще мало времени с начала процедуры, это может показаться неважным, но могут легко возникнуть осложнения. И в этом случае ошибки обходятся дорого. Сначала вся эта процедура будет казаться сложной. Однако вскоре она станет просто еще одним аспектом вашей жизни. Трудно представить себе, что в недалеком будущем исчезнет потребность в выполнении этой задачи, но об этом трудно судить со всей определенностью. После выполнения процедуры вещи нужно снова распределить по разным группам. Затем их можно разложить по соответствующим им местам. В конечном итоге они опять станут использованными и придется еще раз повторить весь цикл. Однако это занятие — часть повседневной жизни.

- 22.2. Используя систему обозначений DCG, напишите грамматику языка, который полностью аналогичен  $\mathcal{E}_1$ , за исключением того, что требует соблюдения согласования между подлежащим и сказуемым предложения, поэтому не позволяет формировать такие предложения, как “I smells the wumpus”.
- 22.3. Дополните грамматику  $\mathcal{E}_1$  таким образом, чтобы в ней соблюдалось согласование артикла и существительного. Иными словами, обеспечьте, чтобы слово “agents” рассматривалось как словосочетание *NP*, а “agent” и “an agents” — нет.
- 22.4. Опишите основные различия между языком Java (или любым другим языком программирования, с которым вы знакомы) и естественным языком, прокомментировав в каждом случае проблему “понимания”. Рассмотрите такие аспекты, как грамматика, синтаксис, семантика, прагматика, композициональность, контекстная зависимость, лексическая неоднозначность, синтаксическая неоднозначность, разрешение ссылок (включая местоимения), фоновые знания, а самое главное, укажите, что означает “понимание” текста на том и другом языке.
- 22.5. Какие из приведенных ниже соображений послужили причинами для введения квазилогической формы?
  - а) Упрощение написания простых композиционных грамматических правил.
  - б) Расширение выразительности языка семантического представления.
  - в) Получение возможности представлять в краткой форме варианты неоднозначности в области определения кванторов (кроме всего прочего).
  - г) Упрощение задачи устранения семантической неоднозначности.
- 22.6. Определите, какая семантическая интерпретация была бы получена при синтаксическом анализе приведенных ниже предложений с помощью грамматики, описанной в данной главе.
  - а) It is a wumpus.
  - б) The wumpus is dead.
  - в) The wumpus is in [2,2].

Было бы целесообразно предусмотреть в качестве семантической интерпретации для предложения “It is a wumpus” просто  $\exists x \ x \in Wumpuses$ ? Рассмотрите альтернативные предложения, такие как “It was a wumpus”.
- 22.7. Не просматривая повторно текст, приведенный в упр. 22.1, ответьте на перечисленные ниже вопросы.

- a) Каковыми являются указанные четыре этапа?
- б) Какой этап пропущен?
- в) Каковыми являются “вещи”, упомянутые в тексте?
- г) Какого рода ошибка может оказаться дорогостоящей?
- д) Лучше ли в данном случае проделать эту процедуру со слишком большим или слишком малым количеством вещей? Объясните, почему.
- 22.8.** В этом упражнении речь идет о грамматиках для очень простых языков.
- а) Напишите контекстно-свободную грамматику для языка  $a^n b^n$ .
- б) Напишите контекстно-свободную грамматику для языка палиндромов — для множества всех строк, вторая половина которых представляет собой обращение первой половины.
- в) Напишите контекстно-зависимую грамматику для языка дубликатов — для множества всех строк, вторая половина которых является такой же, как и первая половина.

- 22.9.** Рассмотрите предложение “Someone walked slowly to the supermarket” (Некто медленно шел в супермаркет) и следующий словарь:

Pronoun → **someone**

V → **walked**

Adv → **slowly**

Prep → **to**

Det → **the**

Noun → **supermarket**

Какая из трех приведенных ниже грамматик в сочетании с этим словарем вырабатывает указанное предложение? Покажите соответствующее дерево (деревья) синтаксического анализа.

A)	B)	C)
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$
$NP \rightarrow Article\ Noun$	$NP \rightarrow Noun$	$NP \rightarrow Article\ NP$
$VP \rightarrow VP\ PP$	$NP \rightarrow Article\ NP$	$VP \rightarrow Verb\ Adv$
$VP \rightarrow VP\ Adv\ Adv$	$VP \rightarrow Verb\ Vmod$	$Adv \rightarrow Adv\ Adv$
$VP \rightarrow Verb$	$Vmod \rightarrow Adv\ Vmod$	$Adv \rightarrow PP$
$PP \rightarrow Prep\ NP$	$Vmod \rightarrow Adv$	$PP \rightarrow Prep\ NP$
$NP \rightarrow Noun$	$Adv \rightarrow PP$	$NP \rightarrow Noun$
	$PP \rightarrow Prep\ NP$	

Применив каждую из трех приведенных здесь грамматик, запишите три правильных и три неправильных английских предложения, сформированных с помощью этой грамматики. Каждое предложение должно иметь значительные отличия от другого, состоять не меньше чем из шести слов и быть основанным на полностью новом множестве словарных записей (которые вы должны определить). Предложите способы совершенствования каждой грамматики, предотвращающие выработку неправильных английских предложений.

- 22.10.** Реализуйте версию алгоритма диаграммного синтаксического анализа, который возвращает упакованное дерево из всех ребер, охватывающих весь объем входных данных.

- 22.11.** Реализуйте версию алгоритма диаграммного синтаксического анализа, который возвращает упакованное дерево для самого длинного крайнего левого ребра, а если это ребро не охватывает весь объем входных данных, продолжает синтаксический анализ с конца этого ребра. Покажите, почему требуется вызывать процедуру *Predict* перед продолжением синтаксического анализа. Окончательным результатом становится такой список упакованных деревьев, что этот список в целом охватывает входные данные.

- 22.12.** (*Составлено по материалам книги [77].*) Данное упражнение относится к языку, получившему название *Buffalo<sup>n</sup>*, который очень напоминает английский (или по меньшей мере его версию  $\mathcal{E}_0$ ), если не считать того, что единственным словом в его словаре является *buffalo*, которое пишется со строчной буквы (и обозначает в данном контексте “стадо буйволов”, если рассматривается как существительное, а в позиции глагола означает “терроризировать”) или с прописной (и обозначает название города — Буффало). Ниже приведены два предложения на этом языке.

Buffalo buffalo buffalo Buffalo buffalo.

Buffalo Buffalo buffalo buffalo Buffalo buffalo.

Если вы не верите, что это действительно предложения, рассмотрите два английских предложения с соответствующей синтаксической структурой:

Dallas cattle bewilder Denver cattle. (Стадо из Далласа терроризирует стадо из Денвера.)

Chefs London critics admire cook French food. (Шеф-повара, которых обожают лондонские критики, готовят блюда французской кухни.)

Напишите грамматику для языка *Buffalo<sup>n</sup>*. Его лексическими категориями являются имя собственное с обозначением города, имя существительное во множественном числе и (переходный) глагол; кроме того, должно быть предусмотрено одно грамматическое правило для предложения, одно — для глагольного словосочетания и три — для именного словосочетания: существительное во множественном числе, именное словосочетание, которому предшествует в качестве модификатора название города, и именное словосочетание, за которым следует сокращенное относительное предложение. Сокращенным относительным предложением называется такое относительное предложение, в котором исключено относительное местоимение. Кроме того, относительное предложение состоит из субъектного именного словосочетания, за которым следует глагол без объекта. Примером сокращенного относительного предложения является “London critics admire” из приведенного выше примера. Рассчитайте количество возможных вариантов синтаксического анализа для языка *Buffalo<sup>n</sup>* при  $n$ , не превышающем 10. *Дополнительное указание.* Карл де Маркен рассчитал, что количество предложений на языке *Buffalo<sup>n</sup>* с длиной 200 (для грамматики, которую он использовал) равно 121 030 872 213 055 159 681 184 485. Как он это сделал?

- 22.13.** Нарисуйте дерево синтаксического анализа речи для приведенного на с. 1087 рассказа о том, как Джон пообедал в модном ресторане. Воспользуйтесь двумя грамматическими правилами для фрагмента текста *Segment*, указав соответствующее значение отношения связности *CoherenceRelation* для каждого узла. (Результаты синтаксического анализа отдельных предложений показы-

вать не обязательно.) Теперь сделайте то же самое для выбранного вами фрагмента речи, состоящего из 5–10 предложений.

- 22.14.** Мы забыли упомянуть, что текст, приведенный в упр. 22.1, должен быть озаглавлен “*Стирка белья*”. Еще раз прочитайте этот текст и ответьте на вопросы, приведенные в упр. 22.7. Удалось ли вам на этот раз лучше справиться с заданием? Бренсфорд и Джонсон [173] использовали этот текст в эксперименте, проводимом под лучшим контролем, и обнаружили, что для его понимания очень важен заголовок. Какие выводы вы можете сделать по проблеме совершенствования речи?

# 23 ВЕРОЯТНОСТНАЯ ОБРАБОТКА ЛИНГВИСТИЧЕСКОЙ ИНФОРМАЦИИ

*В данной главе показано, как можно использовать простые языковые модели, прошедшие статистическое обучение, для обработки коллекций, состоящих из миллионов слов, а не просто отдельных предложений.*

В главе 22 было показано, что агент может взаимодействовать с другим агентом (человеком или программой), используя фрагменты текста на естественном языке. Для полного извлечения смысла фрагментов речи необходимо проводить всесторонний синтаксический и семантический анализ фрагментов речи, а такая возможность возникает благодаря тому, что эти фрагменты речи невелики и относятся только к ограниченной проблемной области.

В данной главе рассматривается подход к обеспечению понимания языка, основанный на использовании **совокупностей текстов**. Совокупностью текстов (*corpus*, во множественном числе — *corpora*) называется большая коллекция текстов, подобная тем миллиардам страниц, из которых состоит World Wide Web. Эти тексты написаны людьми и для людей, а задача программного обеспечения состоит в упрощении поиска нужной информации. В этом подходе предусматривается использование статистики и обучения для получения возможности воспользоваться содержимым совокупности, и в нем обычно применяются вероятностные языковые модели, обучение которых может проводиться с использованием существующих данных и которые проще по сравнению с дополненными грамматиками DCG, описанными в главе 22. При решении большинства подобных задач доступный объем данных превышает тот, который требуется для создания более простой языковой модели. В данной главе рассматриваются три конкретные задачи: информационный поиск (раздел 23.2), извлечение информации (раздел 23.3) и машинный перевод (раздел 23.4). Но вначале в ней представлен обзор вероятностных языковых моделей.

## 23.1. ВЕРОЯТНОСТНЫЕ ЯЗЫКОВЫЕ МОДЕЛИ

В главе 22 описана логическая модель языка; в ней для определения того, относится или не относится к некоторому языку данная строка, использовались грамматики CFG и DCG, а в данном разделе представлено несколько вероятностных моделей. Вероятностные модели имеют целый ряд преимуществ. Обучение этих моделей по имеющимся данным осуществляется очень просто: обучение сводится лишь к подсчету количества вариантов (с учетом определенных допусков на то, что из-за малого размера выборки могут возникать ошибки). Кроме того, эти модели являются более надежными (поскольку они способны принять любую строку, хотя и с низкой вероятностью); они отражают тот факт, что не все 100% говорящих на определенном языке согласны с тем, какие предложения фактически входят в состав языка; кроме того, такие модели могут использоваться для устранения неоднозначности, поскольку для выбора наиболее подходящей интерпретации могут применяться вероятностные законы.

❖ **Вероятностная языковая модель** позволяет определить распределение вероятностей множества строк (которое может быть бесконечно большим). К примерам таких моделей, которые уже рассматривались в данной книге, относятся двух- и трехсловные языковые модели (или модели двух- и трехсловных сочетаний), применявшиеся при распознавании речи (раздел 15.6). В однословной модели (или модели однословных сочетаний) каждому слову в словаре присваивается вероятность  $P(w)$ . В этой модели предполагается, что слова выбираются независимо, поэтому вероятность строки представляет собой произведение вероятностей входящих в нее слов и определяется выражением

$$\prod_i P(w_i).$$

Ниже приведена последовательность из 20 слов, которая была сформирована случайным образом из слов в оригинале данной книги с помощью однословной модели.

logical are as are confusion a may right tries agent goal the was diesel more object then information-gathering search is

В двухсловной модели каждому слову присваивается вероятность  $P(w_i | w_{i-1})$  с учетом предыдущего слова. Часть данных о вероятностях таких двухсловных сочетаний приведена в табл. 15.2. Приведенная ниже случайная последовательность слов сформирована с помощью модели двухсловных сочетаний по материалам оригинала данной книги.

planning purely diagnostic expert systems are very similar computational approach would be represented compactly using tic tac toe a predicate

Вообще говоря, в модели  $n$ -словных сочетаний учитываются предыдущие  $n-1$  слов и присваивается вероятность  $P(w_i | w_{i-(n-1)} \dots w_{i-1})$ . Приведенная ниже случайная последовательность сформирована с помощью модели трехсловных сочетаний по оригиналу данной книги.

planning and scheduling are integrated the success of naive bayes model is just a possible prior source by that time

Даже эти небольшие примеры позволяют понять, что модель трехсловных сочетаний превосходит модель двухсловных сочетаний (а последняя превосходит модель однословных сочетаний) как с точки зрения качества приближенного представления

текста на английском языке, так и с точки зрения успешной аппроксимации изложения темы в книге по искусственному интеллекту. Согласуются и сами модели: в модели трехсловных сочетаний строке, сформированной случайным образом, присваивается вероятность  $10^{-10}$ , в модели двухсловных сочетаний — вероятность  $10^{-29}$ , а в модели однословных сочетаний — вероятность  $10^{-59}$ .

Но оригинал настоящей книги содержит всего лишь полмиллиона слов, поэтому в нем отсутствует достаточный объем данных для выработки качественной модели двухсловных сочетаний, не говоря уже о модели трехсловных сочетаний. Весь словарь оригинала данной книги включает примерно 15 тысяч различных слов, поэтому модель двухсловных сочетаний включает  $15000^2 = 225$  миллионов пар слов. Безусловно, что вероятность появления по меньшей мере 99,8% этих пар будет равна нулю, но сама модель не должна указывать на то, что появление любой из этих пар в тексте невозможно. Поэтому требуется определенный способ **сглаживания** нулевых результатов фактического подсчета количества пар. Простейший способ выполнения этой задачи состоит в использовании так называемого способа **сглаживания с добавлением единицы**: к результатам подсчета количества всех возможных двухсловных сочетаний добавляется единица. Поэтому, если количество слов в текстовой совокупности равно  $N$ , а количество возможных двухсловных сочетаний равно  $B$ , то каждому двухсловному сочетанию с фактическим количеством  $c$  присваивается оценка вероятности  $(c+1) / (N+B)$ . Такой метод позволяет устранить проблему  $n$ -словных сочетаний с нулевой вероятностью, но само предположение, что все результаты подсчета количества должны быть увеличены точно на единицу, является сомнительным и может привести к получению некачественных оценок.

Еще один подход состоит в использовании метода **сглаживания с линейной интерполяцией**, в котором предусматривается объединение моделей трех-, двух- и однословных сочетаний с помощью линейной интерполяции. Оценка вероятности определяется по следующей формуле, с учетом того, что  $c_3+c_2+c_1=1$ :

$$\hat{P}(w_i | w_{i-2}w_{i-1}) = c_3 P(w_i | w_{i-2}w_{i-1}) + c_2 P(w_i | w_{i-1}) + c_1 P(w_i)$$

Параметры  $c_i$  могут быть заранее заданными или полученными путем обучения по алгоритму ЕМ. Существует возможность применения значений  $c_i$ , независимых от количества  $n$ -словных сочетаний, с тем, чтобы можно было присвоить больший вес оценкам вероятностей, полученным на основании больших значений количества.

Один из методов *оценки* языковой модели состоит в следующем. Вначале текстовая совокупность разделяется на обучающую совокупность и контрольную совокупность. Затем определяются параметры модели с помощью обучающих данных. После этого выполняется расчет вероятности, присвоенной контрольной совокупности с помощью данной модели; чем выше эта вероятность, тем лучше. Одним из недостатков этого подхода является то, что вероятность  $P(words)$  при наличии длинных строк становится весьма небольшой; такие малые числовые значения могут вызвать антипереполнение в арифметике с плавающей точкой или просто стать неудобными для чтения. Поэтому вместо вероятности может быть вычислен **показатель связности** (perplexity) модели на контрольной строке слов  $words$  следующим образом:

$$Perplexity(words) = 2^{-\log_2(P(words)) / N}$$

где  $N$  — количество слов  $words$ . Чем ниже показатель связности, тем лучше модель. Модель  $n$ -словных сочетаний, которая присваивает каждому слову вероятность  $1/k$ ,

имеет показатель связности  $k$ ; показатель связности может рассматриваться как средний коэффициент ветвления.

В качестве примера того, для чего может использоваться модель  $n$ -словных сочетаний, рассмотрим задачу **сегментации** — поиска границ между словами в тексте без пробелов. Решением этой задачи обычно приходится заниматься при обработке текстов на японском и китайском языках, в которых отсутствуют пробелы между словами, но авторы полагают, что для большинства читателей более удобным будет пример из английского. Приведенное ниже предложение действительно несложно прочитать любому, кто знает английский язык.

*Itiseasytoreadwordswithoutspaces*

На первый взгляд может показаться, что для решения такой задачи приходится пользоваться всеми знаниями в области синтаксиса, семантики и pragматики английского языка. Но ниже будет показано, что в данном предложении можно легко восстановить пробелы с использованием простой модели однословных сочетаний.

В одной из предыдущих глав было показано, что для решения задачи поиска наиболее вероятной последовательности прохождения через решетку вариантов выбора слова может использоваться уравнение Витерби (15.9). А в листинге 23.1 приведен вариант алгоритма Витерби, специально предназначенный для решения задачи сегментации. Этот алгоритм принимает в качестве входных данных распределение вероятностей однословных сочетаний,  $P(\text{word})$ , и некоторую строку. Затем для каждой позиции  $i$  в данной строке это алгоритм сохраняет в переменной  $\text{best}[i]$  значение вероятности наиболее вероятной строки, которая охватывает участок от начала до позиции  $i$ . Кроме того, в этом алгоритме в переменной  $\text{words}[i]$  сохраняется слово, оканчивающееся в позиции  $i$ , которое получило наибольшую вероятность. После того как по методу динамического программирования будут сформированы массивы  $\text{best}$  и  $\text{words}$ , в алгоритме осуществляется обратный поиск через массив  $\text{words}$  для определения наилучшего пути. В данном случае при использовании модели однословных сочетаний, соответствующей оригиналу этой книги, наиболее приемлемая последовательность слов действительно принимает вид “*It is easy to read words without spaces*” с вероятностью  $10^{-25}$ . Сравнивая отдельные части этого предложения, можно обнаружить, что слово “*easy*” имеет вероятность однословного сочетания  $2.6 \times 10^{-4}$ , а альтернативный вариант его прочтения, “*e as y*”, имеет намного более низкую вероятность,  $9.8 \times 10^{-12}$ , несмотря на тот факт, что слова (точнее, имена переменных) “*e*” и “*y*” довольно часто встречаются в уравнениях данной книги. Аналогичным образом, другая часть этого предложения характеризуется следующими данными:

```

 $P(\text{"without"}) = 0.0004$ 
 $P(\text{"with"}) = 0.005$ 
 $P(\text{"out"}) = 0.0008$ 
 $P(\text{"with out"}) = 0.005 \times 0.0008 = 0.000004$ 

```

**Листинг 23.1. Алгоритм сегментации строки на отдельные слова с помощью уравнения Витерби.** Этот алгоритм восстанавливает наиболее вероятную сегментацию строки на слова после получения строки с удаленными пробелами

---

```

function Viterbi-Segmentation(text, P) returns последовательность
    наиболее подходящих слов sequence и значения вероятностей
    слов этой последовательности

```

**inputs:** *text*, строка символов с удаленными пробелами  
*P*, распределение вероятностей однословных сочетаний  
 среди слов

```

n  $\leftarrow$  Length(text)
words  $\leftarrow$  пустой вектор длины n+1
best  $\leftarrow$  вектор длины n+1, первоначально полностью заполненный
значениями 0.0
best[0]  $\leftarrow$  1.0
/* Заполнить векторы best и words с помощью средств динамического
программирования */
for i = 0 to n do
    for j = 0 to i-1 do
        word  $\leftarrow$  text[j:i]
        w  $\leftarrow$  Length(word)
        if P[word]  $\times$  best[i - w]  $\geq$  best[i] then
            best[i]  $\leftarrow$  P[word]  $\times$  best[i - w]
            words[i]  $\leftarrow$  word
/* Теперь восстановить последовательность слов sequence */
sequence  $\leftarrow$  пустой список
i  $\leftarrow$  n
while i > 0 do
    продвинуть вектор words[i] в начало последовательности sequence
    i  $\leftarrow$  i - Length(words[i])
/* Последовательность наиболее подходящих слов, sequence, и
значения вероятностей слов этой последовательности */
return sequence, best[i]
```

Поэтому слово “without” имеет в 100 раз более высокую вероятность, чем сочетание слов “with out”, согласно применяемой модели однословных сочетаний.

В данном разделе рассматривались модели *n*-элементных сочетаний, элементами которых являются слова, но широкое применение находят также модели *n*-элементных сочетаний, применяемые к другим элементам текста, таким как символы или части речи.

## Вероятностные контекстно-свободные грамматики

В моделях *n*-элементных сочетаний используются статистические данные о совместном появлении элементов в текстовой совокупности, но эти модели не позволяют учитывать грамматические связи на расстояниях, превышающих *n*. В качестве альтернативной языковой модели может служить **вероятностная контекстно-свободная грамматика**, или PCFG<sup>1</sup> (Probabilistic Context-Free Grammar), которая представляет собой такую грамматику CFG, где каждое правило подстановки имеет связанную с ним вероятность. Сумма вероятностей по всем правилам с одной и той же левой частью равна 1. Грамматика PCFG для части грамматики языка  $E_0$  представлена в листинге 23.2.

---

<sup>1</sup> Грамматики PCFG называют также *стохастическими контекстно-свободными грамматиками*, или SCFG (stochastic context-free grammar).

**Листинг 23.2.** Вероятностная контекстно-свободная грамматика (PCFG) и словарь для части грамматики языка  $\mathcal{E}_0$ . Числа в квадратных скобках показывают вероятность того, что вместо символа в левой части правила будет выполнена подстановка правой части соответствующего правила

---

```

 $S \rightarrow NP VP [1.00]$ 
 $NP \rightarrow Pronoun [0.10]$ 
|   Name [0.10]
|   Noun [0.20]
|   Article Noun [0.50]
|   NP PP [0.10]
 $VP \rightarrow Verb [0.60]$ 
|   VP NP [0.20]
|   VP PP [0.20]
 $PP \rightarrow Proposition NP [1.00]$ 
 $Noun \rightarrow \text{breeze} [0.10] \mid \text{wumpus} [0.15] \mid \text{agent} [0.05] \mid \dots$ 
 $Verb \rightarrow \text{sees} [0.15] \mid \text{smells} [0.10] \mid \text{goes} [0.25] \mid \dots$ 
 $Pronoun \rightarrow \text{me} [0.05] \mid \text{you} [0.10] \mid \text{I} [0.25] \mid \text{it} [0.20] \mid \dots$ 
 $Article \rightarrow \text{the} [0.30] \mid \text{a} [0.35] \mid \text{every} [0.05] \mid \dots$ 
 $Proposition \rightarrow \text{to} [0.30] \mid \text{in} [0.25] \mid \text{on} [0.05] \mid \dots$ 

```

---

В модели PCFG вероятность строки,  $P(\text{words})$ , представляет собой сумму вероятностей деревьев синтаксического анализа этой строки. А вероятность данного конкретного дерева представляет собой произведение вероятностей всех правил, на основании которых сформированы узлы этого дерева. На рис. 23.1 показано, как вычислить вероятность некоторого предложения. Такую вероятность можно вычислить, применяя синтаксический анализатор диаграмм CFG для перечисления возможных вариантов синтаксического анализа, а затем складывая полученные вероятности. Но если нас интересует только наиболее вероятный вариант синтаксического анализа, то перебор всех маловероятных вариантов представляет собой бесполезную трату времени. Для эффективного поиска наиболее вероятного варианта синтаксического анализа может использоваться одна из разновидностей алгоритма Виттерби или же какой-то метод поиска по первому наилучшему совпадению (такой как A\*).

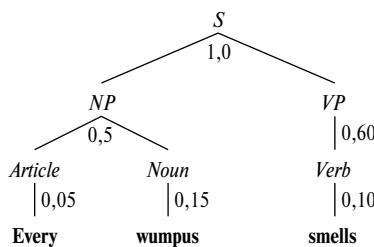


Рис. 23.1. Дерево синтаксического анализа для предложения "Every wumpus smells", в котором показаны вероятности каждого поддерева. Вероятность всего дерева в целом равна  $1.0 \times 0.5 \times 0.05 \times 0.15 \times 0.60 \times 0.10 = 0.000225$ .

Поскольку это дерево является единственным вариантом синтаксического анализа данного предложения, указанное число представляет собой также вероятность этого предложения

Недостатком грамматик PCFG является то, что они — контекстно-свободные. Это означает, что различие между  $P(\text{"eat a banana"})$ , “съешь банан”, и  $P(\text{"eat a bandanna"})$ , “съешь цветной платок”, зависит только от соотношения вероятностей  $P(\text{"banana"})$  и  $P(\text{"bandanna"})$ , а не от вероятностей возникновения отношений между глаголом “eat” и соответствующими объектами. Для того чтобы можно было учитывать связи такого рода, нам потребуется контекстно-зависимая модель определенного типа наподобие **лексикализованной грамматики PCFG**, в которой определенную роль в оценке вероятности соответствующего словосочетания может играть голова<sup>2</sup> этого словосочетания. При наличии достаточного объема обучающих данных может быть получено правило для  $VP \rightarrow VP\ NP$ , обусловленное наличием головы входящего в него словосочетания  $VP$  (“eat”) и головы словосочетания  $NP$  (“banana”). Таким образом, лексикализованные грамматики PCFG позволяют учитывать некоторые ограничения на совместное вхождение элементов в моделях  $n$ -элементных сочетаний, наряду с грамматическими ограничениями моделей CFG.

Еще один недостаток состоит в том, что грамматики PCFG обнаруживают слишком заметное предпочтение по отношению к более коротким предложениям. В такой текстовой совокупности, как архив журнала *Wall Street Journal*, средняя длина предложения составляет около 25 слов. Но обычно грамматика PCFG в конечном итоге присваивает гораздо более высокую вероятность таким правилам, как  $S \rightarrow NP\ VP$ ,  $NP \rightarrow Pronoun$  и  $VP \rightarrow Verb$ . Это означает, что грамматика PCFG присваивает весьма высокую вероятность многим коротким предложениям, таким как “He slept” (Он спал), тогда как в указанном журнале с большей вероятностью встречаются предложения наподобие следующего: “It has been reported by a reliable government source that the allegation that he slept is credible” (Из надежного правительственного источника поступило сообщение, согласно которому заявление о том, что он спал, заслуживает доверия). Создается впечатление, что словосочетания в этом журнале не являются контекстно-свободными; вместо этого его авторы оценивают допустимую ожидаемую длину предложения и используют полученную оценку в качестве мягкого глобального ограничения на структуру составляемых ими предложений. Такой подход к написанию текста трудно отразить в грамматике PCFG.

### Определение с помощью обучения вероятностей для грамматики PCFG

При создании любой грамматики PCFG приходится преодолевать все сложности, связанные с формированием грамматики CFG, и наряду с этим решать проблему задания вероятностей для каждого правила. Такие обстоятельства наводят на мысль, что может оказаться более приемлемым подход, предусматривающий определение грамматики по имеющимся данным с помощью **обучения**, чем подход, основанный на инженерии знаний. Как и в случае распознавания речи, могут применяться данные двух типов — прошедшие и не прошедшие синтаксический анализ. Задача много упрощается, если данные уже преобразованы в деревья с помощью синтаксического анализа лингвистами (или по меньшей мере носителями соответствующего естественного языка, прошедшими специальное обучение). Создание подобной тек-

<sup>2</sup> Головой словосочетания называется самое важное слово, например существительное в именном словосочетании.

стовой совокупности требует больших капиталовложений, и в настоящее время самые крупные из таких совокупностей содержат “всего лишь” около миллиона слов. А если имеется некоторая совокупность деревьев, то появляется возможность создать грамматику PCFG путем подсчета (и сглаживания). Для этого достаточно просмотреть все узлы, в которых корневым является каждый нетерминальный символ, и создать правило для каждой отдельной комбинации дочерних элементов в этих узлах. Например, если какой-то символ  $NP$  появляется 100 тысяч раз и имеется 20 тысяч экземпляров  $NP$  со списком дочерних элементов  $[NP, PP]$ , то создается правило

$$NP \rightarrow NP \ PP \ [0.20]$$

Если же текст не подвергнут синтаксическому анализу, то задача значительно усложняется. Это прежде всего связано с тем, что фактически приходится сталкиваться с двумя разными проблемами — определение с помощью обучения структуры грамматических правил и определение с помощью обучения вероятностей, связанных с каждым правилом (с аналогичным различием приходится сталкиваться при определении с помощью обучения параметров нейронных или байесовских сетей).

На данный момент примем предположение, что структура правил известна и предпринимается лишь попытка определить вероятности с помощью обучения. Для этого может использоваться подход на основе алгоритма ожидания–максимизации (expectation–maximization — EM), как и при обучении моделей НММ. В процессе обучения мы будем пытаться определить такие параметры, как вероятности правил. Скрытыми переменными являются деревья синтаксического анализа, поскольку неизвестно, действительно ли строка слов  $w_1 \dots w_j$  сформирована с помощью правила  $X \rightarrow \alpha$ . На этапе  $E$  оценивается вероятность того, что каждая подпоследовательность сформирована с помощью каждого отдельного правила. Затем на этапе  $M$  оценивается вероятность каждого правила. Весь этот процесс вычисления может осуществляться в режиме динамического программирования с помощью алгоритма, называемого **внутренним–внешним алгоритмом**, по аналогии с прямым–обратным алгоритмом, применяемым для моделей НММ.

На первый взгляд причины продуктивного функционирования внутренне–внешнего алгоритма кажутся непостижимыми, поскольку он позволяет успешно сформировать логическим путем грамматику на основании текста, не прошедшего синтаксический анализ. Но этот алгоритм имеет несколько недостатков. Во-первых, он действует медленно; этот алгоритм характеризуется временными затратами  $O(n^3 t^3)$ , где  $n$  — количество слов в предложении;  $t$  — количество нетерминальных символов. Во-вторых, пространство вероятностных присваиваний очень велико и практика показала, что при использовании этого алгоритма приходится сталкиваться с серьезной проблемой, связанной с тем, что он не выходит из локальных максимумов. Вместо него могут быть опробованы такие альтернативные варианты, как эмуляция отжига, за счет еще большего увеличения объема вычислений. В-третьих, варианты синтаксического анализа, присвоенные с помощью полученных в результате грамматик, часто трудно понять, а лингвисты находят их неудовлетворительными. В результате этого задача комбинирования знаний, представленных с помощью способов, приемлемых для человека, с данными, которые получены с помощью автоматизированного индуктивного логического вывода, становится затруднительной.

### Определение с помощью обучения структуры правил для грамматики PCFG

Теперь предположим, что структура грамматических правил неизвестна. В таком случае сразу же возникает проблема, связанная с тем, что пространство возможных множеств правил является бесконечным, поэтому неизвестно, какое количество правил необходимо предусмотреть и какую длину должно иметь каждое правило. Один из способов решения этой проблемы состоит в том, чтобы организовать составление грамматики с помощью обучения в **нормальной форме Хомского**; это означает, что каждое правило должно находиться в одной из следующих двух форм:

$$\begin{aligned} X &\rightarrow Y \ Z \\ X &\rightarrow \mathbf{t} \end{aligned}$$

где  $X$ ,  $Y$  и  $Z$  — нетерминальные символы;  $\mathbf{t}$  — терминальный символ. В виде грамматики в нормальной форме Хомского, которая распознает точно такой же язык, может быть представлена любая контекстно-свободная грамматика. В таком случае появляется возможность принять произвольное ограничение, согласно которому количество нетерминальных символов будет равно  $n$ , и тем самым будет получено  $n^3 + nv$  правил, где  $v$  — количество терминальных символов. Но практика показала, что такой подход является эффективным только применительно к небольшим грамматикам. Предложен также альтернативный подход, называемый **слиянием байесовских моделей**, аналогичный подходу с применением модели Sequitur (раздел 22.8). В этом подходе предусматривается формирование на первом этапе локальных моделей (грамматик) для каждого предложения, а затем использование минимальной длины описания для слияния моделей.

## 23.2. ИНФОРМАЦИОННЫЙ ПОИСК

**Информационный поиск** — это задача поиска документов, отвечающих потребностям пользователя в информации. Наиболее широко известными примерами систем информационного поиска являются поисковые машины World Wide Web. Пользователь Web может ввести в приглашении поисковой машины такой запрос, как [AI book], и получить список подходящих страниц. В данном разделе показано, как создаются подобные системы. Для систем информационного поиска (называемых сокращенно системами ИП) применяются перечисленные ниже характеристики.

1. Определение коллекции документов. В каждой системе должно быть принято определенное решение о том, что рассматривается в ней как документ — отдельный абзац, страница или многостраничный текст.
2. Способ формулировки **запроса на языке запросов**. Запрос указывает, какая информация требуется пользователю. Язык запросов может предусматривать лишь возможность составления списка слов, такого как [AI book], или может позволять задавать сочетание слов, которые должны быть расположены близко друг от друга, как в запросе ["AI book"]; он может содержать логические операторы, как в запросе [AI AND book]; а также включать операторы, отличные от логических, как в запросе [AI NEAR book] или [AI book SITE:www.aaai.org].

3. **Результирующий набор.** Таковым является подмножество документов, которые система информационного поиска определяет как **релевантные** данному запросу. Под словом *релевантный* подразумевается вероятно полезный (согласно конкретным информационным потребностям, сформулированным в запросе) для того лица, которое сформулировало запрос.
4. Способ **представления** результирующего набора. Он может быть настолько простым, как ранжированный список названий документов, или настолько сложным, как вращающаяся цветная карта результирующего набора, спроектированная на трехмерное пространство.

После чтения предыдущей главы могло сложиться впечатление, что систему информационного поиска возможно создать, преобразовав с помощью синтаксического анализа коллекцию документов в базу знаний, состоящую из логических высказываний, после чего в ней будет выполняться синтаксический анализ каждого запроса и поиск ответа в базе знаний с помощью предиката Ask. Но, к сожалению, еще никому не удалось создать крупномасштабную систему информационного поиска таким образом. Дело в том, что составить словарь и грамматику, которые охватывают большую коллекцию документов, слишком сложно, поэтому во всех системах информационного поиска используются более простые языковые модели.

Самые ранние системы информационного поиска действовали на основе **булевой модели ключевых слов**. Каждое слово в коллекции документов рассматривалось как булева характеристика, которая является истинной применительно к данному документу, если соответствующее слово встречается в документе, и ложной в противном случае. Поэтому характеристика “поиск” является истинной для текущей главы, но ложной для главы 15. В таком случае язык запросов представляет собой язык булевых выражений, заданных на характеристиках. Документ считается релевантным, только если соответствующее выражение принимает истинное значение. Например, запрос [информация AND поиск] принимает истинное значение для текущей главы и ложное для главы 15.

Преимуществом такой модели является то, что ее несложно описать и реализовать. Но она имеет некоторые недостатки. Во-первых, степень релевантности документа измеряется одним битом, поэтому отсутствуют руководящие данные, на основании которых можно было бы упорядочить релевантные документы для презентации. Во-вторых, булевые выражения могут оказаться непривычными для пользователей, не являющимися программистами или логиками. В-третьих, задача формулировки поддающегося запроса может оказаться сложной даже для квалифицированного пользователя. Предположим, что предпринимается попытка выполнить запрос [информация AND поиск AND модели AND оптимизация], что приводит к получению пустого результирующего набора. После этого осуществляется попытка выполнить запрос [информация OR поиск OR модели OR оптимизация], но если он возвращает слишком большой объем результатов, то нелегко определить, какую попытку следует предпринять после этого.

В большинстве систем информационного поиска используются модели, основанные на статистических сведениях о количестве слов (а иногда и другие характеристики низкого уровня). В этой главе будет описана вероятностная инфраструктура, которая хорошо согласуется с описанными ранее языковыми моделями. Основная идея состоит в том, что после формулировки некоторого запроса требуется

найти документы, которые с наибольшей вероятностью будут релевантными по отношению к нему. Иными словами, необходимо вычислить следующее значение вероятности:

$$P(R=\text{true} | D, Q)$$

где  $D$  — документ;  $Q$  — запрос;  $R$  — булева случайная переменная, обозначающая релевантность. После получения этого значения можно применить принцип ранжирования вероятностей, который указывает, что если результирующий набор должен быть представлен в виде упорядоченного списка, это следует сделать в порядке уменьшения вероятности релевантности.

Существует несколько способов декомпозиции совместного распределения  $P(R=\text{true} | D, Q)$ . В настоящей главе будет описан подход, известный под названием **языкового моделирования**, в котором предусматривается получение оценки языковой модели для каждого документа, а затем вычисление для каждого запроса вероятности этого запроса с учетом языковой модели документа. Используя  $r$  для обозначения выражения  $R=\text{true}$ , можно перезаписать приведенное выше определение вероятности следующим образом:

$$\begin{aligned} P(r | D, Q) &= P(D, Q | r) P(r) / P(D, Q) && \text{(согласно правилу Байеса)} \\ &= P(Q | D, r) P(D | r) P(r) / P(D, Q) && \text{(согласно цепному правилу)} \\ &= \alpha P(Q | D, r) P(r | D) / P(D, Q) && \text{(согласно правилу Байеса,} \\ &&& \text{для фиксированного } D) \end{aligned}$$

Как уже было сказано, может быть предпринята попытка максимизировать значение  $P(r | D, Q)$ , но равным образом можно максимизировать отношение вероятностей  $P(r | D, Q) / P(\neg r | D, Q)$ . Это означает, что ранжирование документов может осуществляться на основе следующей оценки:

$$\frac{P(r | D, Q)}{P(\neg r | D, Q)} = \frac{P(Q | D, r)}{P(Q | D, \neg r)} \cdot \frac{P(r | D)}{P(\neg r | D)}$$

Преимущество такого подхода состоит в том, что из процедуры вычисления устриается терм  $P(D, Q)$ . Теперь примем предположение, что в случае нерелевантных документов каждый документ является независимым по отношению к запросу. Иными словами, если какой-то документ нерелевантен по отношению к запросу, то получение информации о существовании этого документа не позволит определить, в чем состоит сам запрос. Это предположение может быть выражено с помощью такой формулы:

$$P(D, Q | \neg r) = P(D | \neg r) P(Q | \neg r)$$

На основании этого предположения получим следующее:

$$\frac{P(r | D, Q)}{P(\neg r | D, Q)} = P(Q | D, r) \times \frac{P(r | D)}{P(\neg r | D)}$$

Коэффициент  $P(r | D) / P(\neg r | D)$  измеряет независимую от запроса вероятность того, что документ является релевантным. Таким образом, этот коэффициент представляет собой меру качества документа; некоторые документы с большей вероятностью будут релевантными по отношению к любому запросу, поскольку сами эти документы имеют изначально высокое качество. Применительно к статьям для академических журналов качество можно оценить на основании количества упоминаний об этих статьях в других источниках, а для оценки Web-страниц можно использовать

количество гиперссылок на ту или иную страницу. В каждом из этих случаев можно присвоить больший вес адресатам ссылок, характеризующимся высоким качеством. Одним из факторов оценки релевантности документа, независимой от запроса, может также служить продолжительность существования этого документа.

Первый коэффициент,  $P(Q|D, r)$ , представляет собой вероятность запроса с учетом релевантного документа. Для оценки этой вероятности необходимо выбрать языковую модель, описывающую то, как связаны запросы с релевантными документами. Один из широко распространенных подходов состоит в том, что документы представляются с помощью модели однословных сочетаний. В проблематике информационного поиска она известна также под названием модели **мультимножества слов**, поскольку в ней учитывается только частота появления каждого слова в документе, а не их порядок. При использовании такой модели следующие (очень короткие) примеры документов рассматриваются как идентичные: “man bites dog” (человек кусает собаку) и “dog bites man” (собака кусает человека). Очевидно, что эти документы имеют разный смысл, но верно также то, что оба они являются релевантными по отношению к запросам о собаках и укусах. Теперь, чтобы рассчитать вероятность запроса при наличии релевантного документа, достаточно просто перемножить вероятности слов в запросе, руководствуясь моделью однословных сочетаний данного документа. В этом и состоит **наивная байесовская** модель данного запроса. Используя  $Q_j$  для обозначения  $j$ -го слова в запросе, получим следующее:

$$P(Q|D, r) = \prod_j P(Q_j|D, r)$$

Это соотношение позволяет ввести такое упрощение:

$$\frac{P(r|D, Q)}{P(\neg r|D, Q)} = \prod_j P(Q_j|D, r) \cdot \frac{P(r|D)}{P(\neg r|D)}$$

Наконец, мы получили возможность применить эти математические модели к некоторому примеру. В табл. 23.1 приведены статистические данные по количеству однословных сочетаний применительно к словам в запросе [Bayes information retrieval model], выполняемом на коллекции документов, состоящей из пяти отдельных глав оригинала настоящей книги. Предполагается, что эти главы имеют одинаковое качество, поэтому требуется лишь вычислить вероятность запроса применительно к данному документу, для каждого документа. Такая процедура выполнена дважды, причем в первый раз использовалось выражение оценки несглаженного максимального правдоподобия  $D_i$ , а во второй раз — модель  $D_i'$  со слаживанием путем добавления единицы. Можно было бы предположить, что текущая глава должна получить наивысший ранг применительно к этому запросу, и в действительности были получены такие данные при использовании в каждой модели.

Преимуществом слаженной модели является то, что она менее восприимчива к шуму и позволяет присвоить ненулевую вероятность релевантности документу, не содержащему все слова запроса. А преимуществом несглаженной модели является то, что она позволяет проще выполнить вычисления применительно к коллекциям с многочисленными документами, поскольку после создания индекса, где указано, в каких документах упоминается каждое слово, появляется возможность быстро формировать результатирующий набор путем применения операции пересечения

к этим спискам, после чего остается вычислить  $P(Q|D_1)$  только для документов, входящих в полученное пересечение, а не для каждого документа.

**Таблица 23.1.** Вероятностная модель информационного поиска для запроса [Bayes information retrieval model], применяемого к коллекции документов, состоящей из пяти глав оригинала настоящей книги. В этой таблице указано количество слов, относящееся к каждой паре “документ–слово”, и общее количество слов  $N$  для каждого документа. Используются две модели документа ( $D_i$  — это несглаженная модель однословных сочетаний для  $i$ -го документа;  $D_i'$  — та же модель со сглаживанием путем добавления единицы) и вычисляется вероятность запроса применительно к каждому документу для обеих моделей. Очевидно, что текущая глава (глава 23) имеет наивысшие показатели при использовании любой модели, поскольку в ней появление искомых слов имеет в 200 раз более высокую вероятность по сравнению с любой другой главой

Слова	Запрос	Глава 1	Глава 13	Глава 15	Глава 22	Глава 23
Bayes	1	5	32	38	0	7
information	1	15	18	8	12	39
retrieval	1	1	1	0	0	17
model	1	9	7	160	9	63
$N$	4	14680	10941	18186	16397	12574
$P(Q D_i, r)$		$1.5 \times 10^{-14}$	$2.8 \times 10^{-13}$	0	0	$1.2 \times 10^{-11}$
$P(Q D_i', r)$		$4.1 \times 10^{-14}$	$7.0 \times 10^{-13}$	$5.2 \times 10^{-13}$	$1.7 \times 10^{-15}$	$1.5 \times 10^{-11}$

### Сравнительный анализ систем информационного поиска

Важная проблема состоит в том, как оценить показатели работы рассматриваемой системы информационного поиска. Проведем эксперимент, в котором системе предъявляется ряд запросов, а результирующие наборы оцениваются с учетом суждений людей о релевантности полученных результатов. По традиции при такой оценке применяются два критерия: полнота выборки и точность. Сформулируем определения этих критериев с помощью примера. Предположим, что некоторая система информационного поиска возвратила результирующий набор, относящийся к одному запросу, применительно к которому известно, какие документы являются и не являются релевантными, из совокупности в 100 документов. Количество документов в каждой категории приведено в табл. 23.2.

**Таблица 23.2. Количество документов в каждой категории**

	В результирующем наборе	Не в результирующем наборе
Релевантный	30	20
Не релевантный	10	40

Показатель **точности** измеряет долю документов в результирующем наборе, которые действительно являются релевантными. В данном примере точность составляет  $30/(30+10)=0,75$ . Относительное количество ложных положительных оценок равно  $1-0,75=0,25$ . Показатель **полноты выборки** измеряет долю всех релевантных документов в коллекции, которые находятся в результирующем наборе. В данном примере полнота выборки составляет  $30/(30+20)=0,60$ . Относительное количество ложных отрицательных оценок равно  $1-0,60=0,40$ . Вычисление показателя полноты выборки в очень большой коллекции документов, такой как World Wide Web, стано-

вится сложным, поскольку отсутствует удобный способ проверки каждой страницы в Web на релевантность. Самое лучшее решение, которое может быть принято в данном случае, состоит в том, чтобы оценивать полноту выборки путем исследования определенной части документов или совсем игнорировать показатель полноты выборки и оценивать коллекцию документов только по показателю точности.

В некоторых системах может происходить потеря точности из-за увеличения полноты выборки. В крайнем случае в системе, которая возвращает в составе результирующего набора каждый документ из коллекции документов, гарантированно достигается полнота выборки, равная 100%, но точность становится низкой. Еще один вариант состоит в том, что система может возвращать единственный документ и показывать низкую полноту выборки, но достигать высокой вероятности получения 100%-ной точности. Один из способов достижения компромисса между точностью и полнотой выборки состоит в использовании **кривой ROC**. Аббревиатура “ROC” сокращенно обозначает показатель “рабочая характеристика приемника” (receiver operating characteristic), который требует дополнительных пояснений. Он представляет собой график, на котором относительное количество ложных отрицательных оценок измеряется по оси  $y$ , а относительное количество положительных оценок измеряется по оси  $x$ , что позволяет находить различные точки компромиссов. Площадь под этой кривой представляет собой суммарную оценку эффективности системы информационного поиска.

Показатели полноты выборки и точности были определены в то время, когда задачи информационного поиска решались главным образом библиотекарями, которые были заинтересованы в получении исчерпывающих, научно обоснованных результатов. В настоящее время большинство запросов (количество которых измеряется сотнями миллионов в сутки) выполняется пользователями Internet, которых в меньшей степени интересует исчерпывающая полнота ответов и требуется лишь немедленно найти ответ. Для таких пользователей одним из наиболее приемлемых критерии является средний **обратный ранг** первого релевантного результата. Это означает, что если первый результат, полученный системой, является релевантным, он получает применительно к данному запросу оценку 1, а если первые два результата не релевантны, а третий является таковым, он получает оценку 1/3. Еще одним критерием служит **время ожидания ответа**, который позволяет измерить продолжительность времени, требуемую для поиска желаемого ответа на поставленный пользователем вопрос. Этот показатель лучше оценивает те характеристики систем информационного поиска, которые действительно хотелось бы точно измерить, но обладает одним недостатком, связанным с тем, что для проведения каждого нового эксперимента приходится привлекать новую партию испытуемых субъектов — людей.

## Совершенствование информационного поиска

В модели однословных сочетаний все слова рассматриваются как полностью независимые, но носителям языка известно, что некоторые слова обладают определенными связями, например, слово “couch” (кушетка) тесно связано со словами “couches” и “sofa”. Во многих системах информационного поиска предпринимаются попытки учитывать подобные корреляции.

Например, если запрос сформулирован как [couch], то исключение из результирующего набора таких документов, в которых упоминаются слова “COUCH” или

“couches”, но не “couch”, было бы неправильным. В большинстве систем информационного поиска используются средства **приведения к нижнему регистру**, с помощью которых слово “COUCH” преобразуется в “couch”, а во многих дополнительно применяется алгоритм **выделения основы**, позволяющий преобразовать слово “couches” в основную форму “couch”. Применение указанных средств обычно позволяет добиться небольшого увеличения полноты выборки (для английского языка такое увеличение составляет порядка 2%). Но использование таких средств может привести к снижению точности. Например, после преобразования слова “stocking” в “stock” с помощью выделения основы обычно снижается точность применительно к запросам, относящимся либо к чулочно-носочным изделиям, либо к финансовым инструментам, хотя и может увеличить полноту выборки применительно к запросам о ведении домашнего хозяйства. Алгоритмы выделения основы, действующие с помощью фиксированных правил (например, правил, предусматривающих удаление суффикса “-ing”), не позволяют предотвратить возникновение этой проблемы, но новейшие алгоритмы, действующие на базе словаря (в которых суффикс “-ing” не удаляется, если слово с этим суффиксом имеется в словаре), позволяют решить эту проблему. Применение средств выделения основы в английском языке не позволяет добиться существенных результатов, но играет более важную роль в других языках. Например, в тексте на немецком языке нередко можно встретить слова наподобие “Lebensversicherungsgesellschaftsangestellter” (служащий компании страхования жизни). В таких языках, как финский, турецкий, инупик и юпик, имеются рекурсивные морфологические правила, которые позволяют в принципе составлять слова неограниченной длины.

Следующий этап состоит в том, что в системе предусматривается распознавание **синонимов**, например, таких, как “sofa” и “couch”. Как и при использовании средств выделения основы, это позволяет добиться небольшого увеличения полноты выборки, но при непродуманном использовании этих средств возникает опасность снижения точности. Пользователи, желающие получить информацию о футболисте Тиме Коуче (Tim Couch), вряд ли хотели бы погрузиться в бесконечные объемы сведений о кушетках и диванах. Проблема состоит в том, что “языки не терпят абсолютной синонимии, так же как природа не терпит вакуума” [312]. Это означает, что при появлении в языке двух слов, соответствующих одному и тому же понятию, люди, говорящие на этом языке, совместными усилиями уточняют толкование таких слов для устранения путаницы.

Во многих системах информационного поиска в определенной степени используются **двухсловные сочетания**, но полная вероятностная модель двухсловных сочетаний реализована лишь в немногих системах. Кроме того, для исправления опечаток как в документах, так и в запросах могут использоваться процедуры **коррекции орфографических ошибок**.

В качестве последнего усовершенствования можно указать, что повышение качества функционирования системы информационного поиска достигается также с помощью использования **метаданных** — данных, внешних по отношению к тексту самого документа. К примерам таких данных относятся ключевые слова, подготовленные разработчиком документа, и гипертекстовые ссылки между документами.

## Способы представления результирующих наборов

В соответствии с принципом вероятностного ранжирования должен быть получен результирующий набор и представлен пользователю в виде списка, отсортированного с учетом вероятности релевантности. Такой способ представления имеет смысл, если пользователь заинтересован в поиске всех релевантных документов, проведенном настолько быстро, насколько это возможно. Но он оказывается не совсем приемлемым, поскольку в нем не учитывается полезность. Например, если в коллекции имеются две копии наиболее релевантного документа, то после просмотра первой копии полезность второй, имеющей такую же релевантность, становится равной нулю. Во многих системах информационного поиска имеются механизмы, позволяющие исключать результаты, которые слишком подобны ранее полученным результатам.

Один из наиболее мощных способов повышения производительности системы информационного поиска состоит в обеспечении возможности использовать **отзывы, касающиеся релевантности**. В этих отзывах пользователь указывает, какие документы из первоначального результирующего набора являются релевантными. После этого система может представить второй результирующий набор документов с документами, подобными указанным.

Еще один подход состоит в том, что результирующий набор представляется в виде размеченного дерева, а не упорядоченного списка. С помощью средств **классификации документов** эти результаты оформляются в виде заранее определенной таксономии тем. Например, коллекция новостных сообщений может классифицироваться на “World News” (Зарубежные новости), “Local News” (Местные новости), “Business” (Новости экономики), “Entertainment” (Новости культуры) и “Sports” (Новости спорта). А при использовании средств **кластеризации документов** дерево категорий создается для каждого результирующего набора с нуля. Методы классификации являются приемлемыми, если количество тем в коллекции невелико, а методы кластеризации в большей степени подходят для более широких коллекций, таких как World Wide Web. И в том и в другом случае после выполнения запроса пользователя результирующий набор предъявляется ему в виде папок, составленных по категориям.

Классификация — это задача контролируемого обучения, поэтому для ее решения может применяться любой из методов, описанных в главе 18. Один из широко используемых подходов состоит в формировании деревьев решений. После подготовки обучающего множества документов, обозначенных правильными категориями, может быть сформировано единственное дерево решений, листьям которого поставлены в соответствие документы, принадлежащие к той или иной категории. Такой подход полностью себя оправдывает, если имеется лишь несколько категорий, но при наличии более крупных множеств категорий приходится формировать по одному дереву решений для каждой категории, притом что листья этого дерева обозначают документ как принадлежащий или не принадлежащий к данной категории. Обычно характеристиками, проверяемыми в каждом узле, являются отдельные слова. Например, в одном из узлов дерева решений для категории “Sports” может быть предусмотрена проверка наличия слова “basketball”. Для классификации текстов были опробованы такие средства, как усиленные деревья решений, наивные байесовские модели и машины поддерживающих векторов; во многих случаях точность при использовании булевой классификации находилась в пределах 90–98%.

Кластеризация относится к типу задач неконтролируемого обучения. В разделе 20.3 было показано, как может использоваться алгоритм ЕМ для улучшения начальной оценки кластеризации на основе сочетания гауссовых моделей. Задача кластеризации документов является более сложной, поскольку неизвестно, было ли выполнено формирование данных с помощью правильной гауссовой модели, а также в связи с тем, что приходится действовать в условиях пространства поиска, имеющего намного больше размерностей. Для решения этой задачи был разработан целый ряд подходов.

В методе **агломеративной кластеризации** создается дерево кластеров путем выполнения полной обработки совокупности вплоть до отдельных документов. Отсечение ветвей этого дерева для получения меньшего количества категорий может быть выполнено на любом уровне, но такая операция рассматривается как выходящая за рамки самого алгоритма. На первом этапе каждый документ рассматривается как отдельный кластер. После этого отыскиваются два кластера, наиболее близкие друг к другу согласно определенному критерию расстояния, и эти два кластера сливаются в один. Такой процесс повторяется до тех пор, пока не остается только один кластер. Критерием расстояния между двумя документами является некоторый критерий, измеряющий совпадение слов в этих документах. Например, документ может быть представлен как вектор количества слов, а само расстояние определено как евклидово расстояние между двумя векторами. Критерием расстояния между двумя кластерами может служить расстояние до середины кластера или может учитываться среднее расстояние между элементами кластеров. Метод агломеративной кластеризации требует затрат времени, пропорциональных  $O(n^2)$ , где  $n$  — количество документов.

В методе **кластеризации по  $k$  средним** создается плоское множество, состоящее точно из  $k$  категорий. Этот метод действует, как описано ниже.

1. Случайным образом осуществляется выборка  $k$  документов для представления  $k$  категорий.
2. Каждый документ обозначается как принадлежащий к ближайшей категории.
3. Вычисляется среднее каждого кластера и используются  $k$  средних для представления новых значений  $k$  категорий.
4. Этапы 2) и 3) повторяются до тех пор, пока алгоритм не сходится.

Для метода кластеризации по  $k$  средним требуются затраты времени, пропорциональные  $O(n)$ , в чем состоит одно из его преимуществ над агломеративной кластеризацией. Но в литературе часто приходится встречать сообщение о том, что этот метод является менее точным по сравнению с агломеративной кластеризацией, хотя некоторые исследователи сообщают, что он позволяет добиться почти таких же высоких показателей [1460].

Но независимо от применяемого алгоритма кластеризации требуется решить еще одну задачу, прежде чем результаты кластеризации можно будет использовать для представления результирующего набора, — найти удобный способ описания кластера. При использовании метода классификации имена категорий уже определены (например, “Earnings” — доходы), но при кластеризации имена категорий приходится изобретать заново. Один из способов выполнения этой задачи состоит в подборе списка слов, которые являются представительными для этого кластера. Еще один вариант состоит в применении названий одного или нескольких документов, близких к центру кластера.

## Создание систем информационного поиска

До сих пор в этой главе было приведено описание работы систем информационного поиска в общих чертах, но не показано, как добиться эффективного функционирования этих систем для того, чтобы машины поиска Web могли возвращать искомые результаты обработки коллекции, состоящей из многих миллиардов страниц, за десятые доли секунды. Двумя основными структурами данных любой системы информационного поиска являются лексикон, содержащий списки всех слов в рассматриваемой коллекции документов, и инвертированный индекс, в котором перечислены все места, где каждое слово встречается в коллекции документов.

**Лексиконом** называется структура данных, которая поддерживает одну важную операцию: после получения определенного слова она возвращает данные о том, в каком месте инвертированного индекса хранятся экземпляры этого слова. В некоторых версиях систем информационного поиска эта структура возвращает также данные об общем количестве документов, содержащих искомое слово. Лексикон должен быть реализован с использованием хэш-таблицы или аналогичной структуры данных, которая обеспечивает быстрое выполнение этой операции поиска. Иногда в лексикон не включают ряд широко распространенных слов, имеющих малое информационное содержание. Эти слова, называемыми **запретными словами** (“the”, “of”, “to”, “be”, “a” и т.д.), только занимают место в индексе, но не увеличивают ценность результата. Единственным резонным основанием для включения их в лексикон может служить вариант, в котором лексикон используется для поддержки фразовых запросов, — индекс, содержащий запретные слова, необходим для эффективной выборки результатов для таких запросов, как “to be or not to be”.

**Инвертированный индекс<sup>3</sup>**, подобно индексу (предметному указателю), приведенному в конце данной книги, состоит из множества **списков позиций** — обозначений тех мест, где встречается каждое слово. Применительно к булевой модели ключевых слов список позиций представляет собой список документов. А список позиций, применяемый в модели однословных сочетаний, представляет собой список пар (документ, количество). Для обеспечения поддержки фразового поиска список позиций должен также включать обозначения позиций в каждом документе, где встречается каждое слово.

Если запрос состоит из одного слова (а такая ситуация встречается в 26% случаев, согласно [1411]), его обработка происходит очень быстро. Для этого выполняется единственная операция поиска в лексиконе для получения адреса списка позиций, а затем создается пустая очередь по приоритету. В дальнейшем происходит обработка списка позиций одновременно по одному документу и проверка количества экземпляров искомого слова в документе. Если очередь по приоритету имеет меньше, чем  $R$  элементов (где  $R$  — размер желаемого результирующего набора), то пара (документ, количество) добавляется к очереди. В противном случае, если количество экземпляров искомого слова больше по сравнению с соответствующими данными элемента с наименьшими показателями в очереди по приоритету, этот элемент

<sup>3</sup> Термин “инвертированный индекс” (inverted index) является избыточным; лучшим термином был бы просто “индекс”. Индекс называют инвертированным потому, что он задает порядок расположения слов, отличный от того порядка, в котором слова расположены в тексте, но таковы все индексы. Тем не менее по традиции в системах информационного поиска применяется термин “инвертированный индекс”.

с наименьшими показателями удаляется из очереди и добавляется новая пара (документ, количество). Таким образом, поиск ответа на запрос требует затрат времени, пропорциональных  $O(H+R\log R)$ , где  $H$  — количество документов в списке позиций. Если запрос состоит из  $n$  слов, требуется выполнить слияние  $n$  списков позиций, для чего требуется затраты времени, равные  $O(nH+R\log R)$ .

В данной главе теоретический обзор средств информационного поиска представлен с использованием вероятностной модели, поскольку эта модель основана на идеях, уже описанных при изложении других тем в настоящей книге. Но в системах информационного поиска, фактически применяемых на практике, чаще всего используется другой подход, называемый **моделью векторного пространства**. Эта модель основана на таком же подходе с использованием мультимножества слов, как и вероятностная модель. Каждый документ представлен в виде вектора частот однословных сочетаний. Запрос также представляется полностью аналогичным образом; например, запрос [Bayes information retrieval model] представляется в виде вектора:

[0, ..., 1, 0, ..., 1, 0, ..., 1, 0, ..., 1, 0, ...]

Применяемая здесь идея состоит в том, что существует по одному измерению для каждого слова в коллекции документов, а запрос получает оценку 0 по каждому измерению, кроме тех четырех, которые фактически присутствуют в запросе. Релевантные документы отбираются путем поиска среди векторов документов именно тех векторов, которые являются ближайшими соседями по отношению к вектору запроса в векторном пространстве. Одним из критериев подобия служит точечное произведение между вектором запроса и вектором документа; чем больше это произведение, тем ближе два вектора. С точки зрения алгебры указанные вычисления обеспечивают получение высоких оценок теми словами, которые часто появляются и в документе, и в запросе. А с точки зрения геометрии точечное произведение между двумя векторами равно косинусу угла между этими векторами; максимизация косинуса угла между двумя векторами (находящимися в одном и том же квадранте) равносильна уменьшению этого угла до нуля.

Это краткое описание далеко не исчерпывает всю проблематику модели векторного пространства. На практике эта модель была развита до такой степени, чтобы в ней можно было учесть целый ряд дополнительных средств, уточнений, исправлений и дополнений. Основная идея ранжирования документов по их подобию в векторном пространстве позволяет внести новые понятия в систему числового ранжирования. Некоторые специалисты утверждают, что вероятностная модель позволила бы выполнять аналогичные манипуляции более научно обоснованным способом, но исследователи в области информационного поиска вряд ли согласятся перейти на другой инструментарий до тех пор, пока не убедятся в явных преимуществах другой модели с точки зрения производительности.

Для того чтобы получить представление о том, с какими масштабами применения средств индексации приходится сталкиваться при решении типичной задачи информационного поиска, рассмотрим стандартную совокупность документов из коллекции TREC (Text REtrieval Conference), состоящую из 750 тысяч документов с общим объемом в 2 Гбайт текста. Лексикон этой коллекции содержит приблизительно 500 тысяч слов, к которым применены операции выделения основы и приведения к нижнему регистру; для хранения этих слов требуется объем памяти от 7 до 10 Мбайт. Инвертированный индекс с парами (документ, количество) занимает 324

Мбайт, хотя и остается возможность применить методы сжатия для сокращения этого объема до 83 Мбайт. Методы сжатия позволяют экономить пространство за счет небольшого увеличения потребностей в обработке. Но если сжатие позволяет держать весь индекс в памяти, а не хранить его на диске, то появляется возможность добиться существенного общего прироста производительности. Для поддержки фразовых запросов требуется увеличение этого объема примерно до 1200 Мбайт не в сжатом виде или до 600 Мбайт со сжатием. Машины поиска Web действуют в масштабах, превышающих примерно в 3000 раз указанные выше. При этом многие из описанных здесь проблем остаются теми же, а поскольку задача оперирования с тегами данных в одном компьютере практически не осуществима, индекс разделяется на  $k$  сегментов и каждой сегмент сохраняется на отдельном компьютере. Запрос передается параллельно на все компьютеры, а затем  $k$  результирующих наборов сливаются в один результирующий набор, который предъявляется пользователю. Кроме того, машины поиска Web вынуждены справляться с тысячами запросов, поступающих в секунду, поэтому для них требуется  $n$  копий  $k$  компьютеров. Со временем значения  $k$  и  $n$  продолжают возрастать.

### 23.3. ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ

Извлечением информации называется процесс создания записей базы данных путем просмотра текста и выявления экземпляров конкретного класса объектов или событий, а также связей между этими объектами и событиями. Может быть предпринята попытка применить такой процесс для извлечения данных об адресах из Web-страниц и внесения в базу данных информации об улице, городе, штате и почтовом коде или извлечения сведений о происходящих штормах из сообщений о погоде и внесения в базу данных информации о температуре, скорости ветра и количестве осадков. Системы извлечения информации занимают промежуточное положение между системами информационного поиска и полными синтаксическими анализаторами текста, поскольку к ним предъявляются более высокие требования, чем просто преобразование документа в мультимножество слов, но меньшие требования по сравнению с полным анализом каждого предложения.

Простейшим типом системы извлечения информации является система, основанная на атрибуатах, поскольку в ней предполагается, что весь текст относится к одному объекту и задача состоит в извлечении атрибутов этого объекта. Например, в разделе 10.5 упоминалась задача извлечения из текста “17in SXGA Monitor for only \$249.99” отношений базы данных, определяемых следующим выражением:

$$\exists m \in ComputerMonitors \wedge Size(m, Inches(17)) \wedge Price(m, \$249.99) \\ \wedge Resolution(m, 1280 \times 1024)$$

Определенная часть этой информации может обрабатываться с помощью регулярных выражений, которые определяют регулярную грамматику, заданную на одной строке текста. Регулярные выражения используются в командах Unix, таких как grep, в языках программирования, таких как Perl, и в текстовых процессорах, таких как Microsoft Word. Подробные сведения о грамматике, применяемой в том или ином инструментальном средстве, в значительной степени различаются, поэтому их лучше всего узнать из соответствующего справочного руководства, но в

табл. 23.3 показано, как сформировать регулярное выражение для выделения данных о ценах в долларах, и продемонстрировано применение общих подвыражений.

**Таблица 23.3. Примеры применения регулярных выражений**

Регулярное выражение	Результат применения
[0-9]	Согласуется с любой цифрой от 0 до 9
[0-9] +	Согласуется с одной или большим количеством цифр
. [0-9] [0-9]	Согласуется с конструкцией, состоящей из точки, за которой следуют две цифры
( . [0-9] [0-9] ) ?	Согласуется с конструкцией, состоящей из точки, за которой следуют две цифры, или с пустой строкой
\$ [0-9] + ( . [0-9] [0-9] ) ?	Согласуется со строкой \$249 . 99, или \$1 . 23, или \$1000000, или ...

Системы извлечения информации на основе атрибутов могут быть созданы в виде ряда регулярных выражений, по одному для каждого атрибута. Если регулярное выражение согласуется с текстом один и только один раз, то существует возможность извлечь часть текста, определяющую значение соответствующего атрибута. Если соответствия не найдены, то больше ничего нельзя сделать, а если регулярное выражение согласуется с текстом в нескольких местах, то нужно применить процесс осуществления выбора между этими согласованиями. Одна из возможных стратегий состоит в том, чтобы для каждого атрибута было предусмотрено несколько регулярных выражений, упорядоченных по приоритетам. Поэтому, например, регулярное выражение с наивысшим приоритетом для выделения цены может предусматривать применение строки “*our price:*”, за которой сразу же следует знак доллара “\$”; если же эта строка не будет обнаружена, можно сразу же перейти к использованию менее надежного регулярного выражения. Еще одна стратегия состоит в том, чтобы найти все согласования и применить определенный способ выбора между ними. Например, можно взять самую низкую цену, которая находится в пределах 50% от самой высокой цены. Это позволить обрабатывать тексты, подобные следующему: “List price \$99.00, special sale price \$78.00, shipping \$3.00”.

На более высоком этапе развития по сравнению с системами извлечения информации на основе атрибутов находятся системы извлечения информации на основе отношений, или реляционные системы, которые позволяют учитывать наличие в тексте информации о более чем одном объекте и отношениях между ними. Таким образом, при обнаружении такими системами текста “\$249 . 99” они должны определить не только цену, но и объект, имеющий эту цену. Типичной системой извлечения информации на основе отношений является система FASTUS, которая применяется для обработки новостных сообщений о корпоративных слияниях и приобретениях. Эта система способна прочитать следующее сообщение:

Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

и сформировать примерно такую запись базы данных:

```
e ∈ JointVentures ∧ Product(e, "golf clubs") ∧ Date(e, "Friday")
  ∧ Entity(e, "Bridgestone Sports Co") ∧ Entity(e, "a local concern")
  ∧ Entity(e, "a Japanese trading house")
```

Реляционные системы извлечения информации часто создаются на основе ~~и каскадных преобразователей с конечными автоматами~~. Это означает, что они состоят из ряда конечных автоматов (Finite-State Automaton — FSA), где каждый автомат принимает текст в качестве входных данных, преобразует этот текст в другой формат и передает его следующему автомatu. Такой способ обработки является осуществимым, поскольку каждый конечный автомат действует достаточно эффективно, а при совместном использовании они приобретают способность извлекать необходимую информацию. Типичной системой такого типа является FASTUS, которая состоит из конечных автоматов, выполняющих описанные ниже пять этапов обработки.

1. Разбиение на лексемы.
2. Обработка сложных слов.
3. Обработка базовых групп.
4. Обработка сложных фраз.
5. Слияние структур.

Первым этапом обработки системы FASTUS является **разбиение на лексемы**, в котором поток символов сегментируется на лексемы (слова, числа и знаки препинания). Применительно к тексту на английском языке разбиение на лексемы может быть выполнено довольно просто; для этого достаточно лишь следить за разделяющими символы пробелами или знаками препинания. А применительно к тексту на японском языке для разбиения на лексемы требуется вначале выполнить сегментацию, используя нечто вроде алгоритма сегментации Витерби (см. листинг 23.1). Некоторые средства разбиения на лексемы позволяют также обрабатывать такие языки разметки, как HTML, SGML и XML.

На втором этапе обрабатываются **сложные слова**, включая такие словосочетания, как “set up” (настройка) и “joint venture” (совместное предприятие), а также имена собственные, такие как “Prime Minister Tony Blair” и “Bridgestone Sports Co.”. Сложные слова распознаются с использованием сочетания лексических элементов и грамматических правил конечного автомата. Например, название компании может быть распознано с помощью следующего правила:

```
CapitalizedWord+ ("Company" | "Co" | "Inc" | "Ltd")
```

Эти правила необходимо составлять с учетом всех предосторожностей и проверять на полноту и точность. Одна из коммерческих систем распознала словосочетание “Intel Chairman Andy Grove” (Председатель правления Intel Энди Гроув) как обозначение местности, а не имя лица, применив правило в следующей форме:

```
CapitalizedWord+ ("Grove" | "Forest" | "Village" | ...)
```

На третьем этапе выполняется обработка **базовых групп**; под этим подразумеваются именные и глагольные группы. Общая идея состоит в том, чтобы объединить на этом этапе слова в такие элементы, которые можно будет легко обрабатывать на последующих этапах. Именная группа состоит из заглавного существительного, за которым следуют необязательные определители и другие модификаторы. Поскольку именная группа не включает всех сложных конструкций, предусмотренных для именного словосочетания  $NP$  в грамматике  $\mathcal{E}_1$ , не требуются рекурсивные правила контекстно-свободной грамматики — достаточно только использовать правила регу-

лярной грамматики, допустимые для конечных автоматов. Глагольная группа состоит из глагола и присоединенных к нему вспомогательных частиц и наречий, но без прямого и косвенного объекта и пропозициональных предложений. Предложение, приведенное выше в качестве примера, может быть преобразовано на этом этапе в следующую конструкцию:

- |                                       |                                          |
|---------------------------------------|------------------------------------------|
| 1. <i>NG</i> : Bridgestone Sports Co. | 10. <i>NG</i> : a local concern          |
| 2. <i>VG</i> : said                   | 11. <i>CJ</i> : and                      |
| 3. <i>NG</i> : Friday                 | 12. <i>NG</i> : a Japanese trading house |
| 4. <i>NG</i> : it                     | 13. <i>VG</i> : to produce               |
| 5. <i>VG</i> : had set up             | 14. <i>NG</i> : golf clubs               |
| 6. <i>NG</i> : a joint venture        | 15. <i>VG</i> : to be shipped            |
| 7. <i>PR</i> : in                     | 16. <i>PR</i> : to                       |
| 8. <i>NG</i> : Taiwan                 | 17. <i>NG</i> : Japan                    |
| 9. <i>PR</i> : with                   |                                          |

где *NG* обозначает именную группу; *VG* — глагольную группу; *PR* — предлог, *CJ* — союз.

На четвертом этапе базовые группы объединяются в **сложные фразы**. И в этом случае цель состоит в том, чтобы применяемые правила могли быть реализованы с помощью конечного автомата и допускали быструю обработку, а полученный результат сводился к непротиворечивым (или почти непротиворечивым) выходным фразам. В правиле комбинирования одного из типов учитываются события, типичные для рассматриваемой проблемной области. Например, следующее правило отражает один из способов описания процесса формирования совместного предприятия:

```
Company+SetUp JointVenture("with" Company+) ?
```

Этот этап является первым из каскада этапов, в которых полученные выходные данные помещаются в шаблон базы данных, а также выводятся в выходной поток.

На последнем этапе происходит **слияние структур**, которые были сформированы на предыдущем этапе. Если в следующем предложении сказано: “The joint venture will start production in January” (Это совместное предприятие начнет выпускать продукцию в январе), то на данном этапе будет отмечено, что в двух ссылках на совместное предприятие (“joint venture”) упоминается один и тот же объект, и они будут объединены в одну ссылку.

Вообще говоря, средства извлечения информации действуют успешно применительно к ограниченной проблемной области, в которой возможно заранее определить, какие темы будут обсуждаться и в каких терминах будет проходить это обсуждение. Такие средства показали свою применимость для целого ряда проблемных областей, но они не способны заменить полномасштабную обработку текста на естественном языке.

## 23.4. МАШИННЫЙ ПЕРЕВОД

*Машинным переводом* называется автоматический перевод текста с одного естественного языка (исходного) на другой (целевой). Практика показала, что этот процесс может применяться для выполнения целого ряда задач, включая перечисленные ниже.

1. Грубый перевод, цель которого состоит в том, чтобы только определить смысл отрывка текста. В нем допускается наличие грамматически неправильных и неуклюжих предложений, при условии, что смысл этих предложений ясен. Например, в ходе Web-серфинга пользователю часто достаточно получить грубый перевод Web-страницы на иностранном языке. Иногда лицо, владеющее только родным языком, может успешно выполнить последующее редактирование результатов перевода без необходимости читать источник. Такого рода перевод с помощью машины позволяет сэкономить деньги, поскольку тем, кто занимается редактированием текста, полученного с помощью машинного перевода, можно платить меньше, чем тем, кто непосредственно переводит с иностранного языка.
2. Перевод источников с ограниченной тематикой, в котором тема и формат исходного текста строго ограничены. Одним из наиболее удачных примеров является система Taum-Meteo, которая переводит сообщения о погоде с английского языка на французский. Ее работа основана на том, что язык, используемый в сообщениях о погоде, является в высшей степени стилизованным и формализованным.
3. Перевод с предварительным редактированием, в котором исходный документ заранее редактируется перед машинным переводом людьми для того, чтобы он соответствовал ограниченному подмножеству английского или любого другого языка оригинала. Такой подход является особенно экономически эффективным, если есть необходимость перевести один документ на много языков, как в случае распространения юридических документов в Европейском экономическом сообществе или в случае тиражирования инструкций компаниями, которые продают один и тот же продукт во многие страны. Ограниченные языки иногда называют “Caterpillar English” (английским языком Caterpillar), поскольку впервые попытку оформлять свои инструкции в такой форме предприняла корпорация Caterpillar. Компания Xerox определила язык для своих инструкций по техническому обслуживанию, который является достаточно простым для того, чтобы его можно было перевести с помощью машины на языки всех стран, с которыми компания Xerox имеет деловые связи. Дополнительным преимуществом оказалось то, что оригинальные английские инструкции также стали более понятными.
4. Литературный перевод, в котором сохраняются все нюансы исходного текста. В настоящее время эта задача выходит за рамки возможностей машинного перевода.

В качестве примера грубого перевода в табл. 23.4 приведен первый абзац из оригинала данной главы, переведенный на итальянский язык, затем снова на английский с помощью службы перевода Systran.

Задача перевода является сложной, поскольку, вообще говоря, для ее решения требуется глубокое понимание текста, а для этого, в свою очередь, необходимо глубокое понимание ситуации, о которой идет речь. Это утверждение является справедливым применительно даже к очень простым текстам, в частности даже к “текстам”, состоящим из одного слова. Рассмотрим слово “Open” на двери магазина<sup>4</sup>. Оно со-

<sup>4</sup> Этот пример предложил нам Мартин Кэй (Martin Kay).

общает, что в данный момент магазин принимает покупателей. Теперь предположим, что такое же слово “Open” написано на большом плакате рядом с вновь открытым магазином. Оно означает, что магазин уже работает, но люди, прочитавшие это сообщение, нечувствуют себя обманутыми, узнав, что магазин закрыли на ночь, а плакат оставил висеть на стенде. Дело в том, что в этих двух сообщениях одно и то же слово использовалось для передачи разных смысловых значений. С другой стороны, в тех магазинах, где с покупателями общаются на немецком языке, на дверях принято вешать табличку “Offen”, а на соответствующих плакатах, сообщающих об открытии нового магазина, писать слова “Neu Eröffnet”.

**Таблица 23.4. Результаты двух последовательных применений средств грубого машинного перевода текста первого абзаца из оригинала настоящей главы**

Текст на итальянском языке	Текст на английском языке
In capitolo 22 abbiamo visto come un agente potrebbe comunicare con un altro agente (essere umano o software) che usando le espressioni in un linguaggio reciprocamente accordato. Completare sintattico e l'analisi semantica delle espressioni è necessaria da estrarre il significato completo del utterances ed è possibile perché le espressioni sono corte e limitate ad un settore limitato	In chapter 22 we have seen as an agent could communicate with another agent (to be human or software) that using the expressions in a language mutual come to an agreement. Complete syntactic and the semantic analysis of the expressions is necessary to extract the complete meant one of the utterances and is possible because the expressions short and are limited to a dominion

Проблема состоит в том, что в различных языках мир подразделяется на категории по-разному. Например, французское слово “doux” охватывает широкий диапазон смысловых значений, приблизительно соответствующих английским словам “soft”, “sweet” и “gentle”. Аналогичным образом, английское слово “hard” охватывает практически все области применения немецкого слова “hart” (физически стойкий, жесткий), а также области использования слова “schwierig” (трудный). Немецкий глагол “heilen” охватывает все области применения английского существительного “cure” в медицине, а также области применения английского глагола “heal” в качестве транзитивного и нетранзитивного в обычном языке. Поэтому задача представления смысла предложения для перевода сложнее по сравнению с той ситуацией, когда эта задача решается в целях понимания смысла предложения на одном языке. В системе синтаксического анализа текста на одном языке могут использоваться предикаты наподобие *Open(x)*, а в случае перевода язык представления должен обеспечивать проведение более тонких различий, например, с учетом того, что предикат *Open<sub>1</sub>(x)* должен представлять смысл надписи “Offen”, а *Open<sub>2</sub>(x)* — смысл надписи “Neu Eröffnet”. Язык представления, в котором учитываются все различия, необходимые для представления целого ряда языков, называется **промежуточным языком**.

Чтобы выполнить беглый перевод, переводчик (человек или машина) должен прочитать первоначальный текст, понять тему, к которой он относится, и составить соответствующий текст на целевом языке, достаточно качественно описывающий ту же или аналогичную тему. При этом часто приходится брать на себя определенную ответственность. Например, английское слово “you”, обращенное кциальному лицу, может быть переведено на французский язык либо как формальное обращение “vous”, либо как неформальное “tu”. Дело в том, что просто не существует способа, позволяющего перевести обращение “you” на французский язык, не приняв вместе с

тем решения о том, должно ли это обращение быть формальным или неформальным. Переводчики (и машины, и люди) иногда испытывают затруднения, принимая подобные решения.

### Системы машинного перевода

Системы машинного перевода различаются по тому признаку, на каком уровне в них осуществляется анализ текста. В некоторых системах предпринимается попытка полностью проанализировать входной текст вплоть до получения представления на промежуточном языке (как было сделано в главе 22), а затем сформировать из этого представления предложения на целевом языке. Такая задача является сложной, поскольку она включает в качестве подзадачи проблему полного понимания языка, а к этому добавляются все сложности, связанные с применение промежуточного языка. К тому же такой подход является ненадежным, поскольку в случае неудачного завершения анализа не формируются также выходные данные. А его преимуществом является то, что нет таких частей системы, функционирование которых основано на знании одновременно двух языков. Это означает, что может быть создана система с промежуточным языком, позволяющая переводить тексты с одного из  $n$  языков на другой за счет трудозатрат, пропорциональных  $O(n)$ , а не  $O(n^2)$ .

Другие системы основаны на так называемом методе **передачи**. В них имеется база данных, состоящая из правил перевода (или примеров), а перевод осуществляется непосредственно путем согласования текста с правилами (или примерами). Передача может осуществляться на лексическом, синтаксическом или семантическом уровне. Например, строго синтаксическое правило преобразует английское словосочетание [*Adjective Noun*] (*adjective* — имя прилагательное, *noun* — имя существительное) во французское словосочетание [*Noun Adjective*]. А, допустим, смешанное синтаксическое и лексическое правило устанавливает соответствие между французским выражением [ $S_1$  "et puis"  $S_2$ ] и английским [ $S_1$  "and then"  $S_2$ ]. Передача, выполняемая с непосредственным преобразованием одного предложения в другое, известна под названием метода **перевода с помощью памяти**, поскольку она основана на запоминании большого множества пар (английский, французский). Метод передачи является надежным, поскольку он всегда обеспечивает выработку пусть даже каких-то выходных данных и при этом по меньшей мере хотя бы часть полученных слов обязательно оказывается правильной. На рис. 23.2 схематически показаны различные уровни передачи.

### Статистический машинный перевод

В начале 1960-х годов широко распространилось мнение, что компьютеры вскоре смогут без особых проблем переводить с одного естественного языка на другой, в соответствии с тем, что в проекте Тьюринга удалось добиться успешного “перевода” закодированных сообщений на немецком языке в немецкий текст, доступный для восприятия. Но к 1966 году стало ясно, что для беглого перевода требуется понимание смысла сообщений, а для взлома кода — нет.

В последнее десятилетие наметилась тенденция к использованию систем машинного перевода, основанных на статистическом анализе. Безусловно, можно добиться выигрыша благодаря применению статистических данных и четкой вероятностной

модели того, в чем состоит качественный анализ или передача текста, на любом из этапов, показанных на рис. 23.2. Но под понятием “статистического машинного перевода” подразумевается общий подход к решению проблемы перевода, который основан на поиске наиболее вероятного перевода предложения с использованием данных, полученных из двуязычной совокупности текстов. В качестве примера двуязычной совокупности текстов можно назвать **парламентские отчеты**<sup>5</sup>, которые представляют собой протоколы дебатов в парламенте. Двуязычные парламентские отчеты издаются в Канаде, Гонконге и других странах; официальные документы Европейского экономического сообщества издаются на 11 языках; а Организация объединенных наций публикует документы на нескольких языках. Как оказалось, эти материалы представляют собой бесценные ресурсы для статистического машинного перевода.

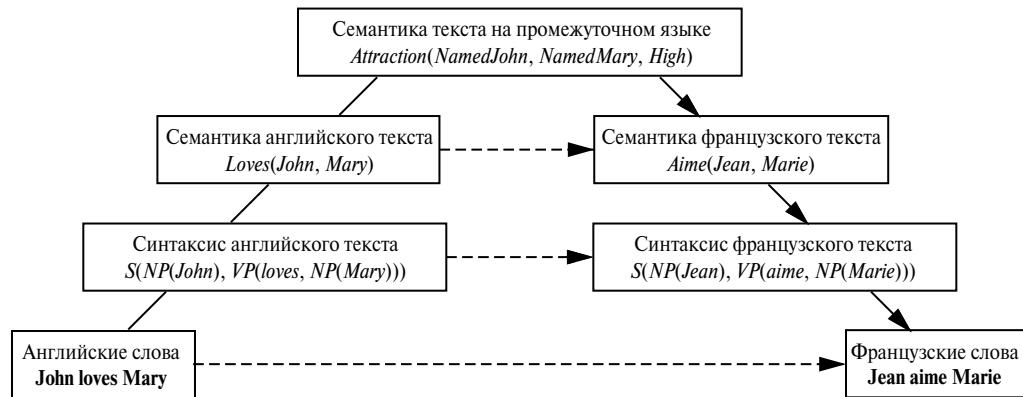


Рис. 23.2. Схематическое изображение вариантов организации систем машинного перевода. Схема начинается с английского текста, показанного в левой нижней части. Система с промежуточным языком следует по сплошным линиям, выполняя синтаксический анализ английского текста и преобразуя его вначале в синтаксическую форму, затем в семантическую форму представления и в форму представления на промежуточном языке, после этого выполняет этапы преобразования в семантическую, синтаксическую и лексическую форму на французском языке. В системе на основе передачи в качестве сокращенных путей используются пунктирные линии. В различных системах передача осуществляется на разных уровнях, причем в некоторых системах она происходит одновременно на нескольких уровнях

Проблему перевода английского предложения  $E$ , скажем, во французское<sup>6</sup> предложение  $F$  можно представить в виде следующего уравнения, предусматривающего применение правила Байеса:

$$\begin{aligned} \underset{F}{\operatorname{argmax}} P(F|E) &= \underset{F}{\operatorname{argmax}} P(E|F) P(F) / P(E) \\ &= \underset{F}{\operatorname{argmax}} P(E|F) P(F) \end{aligned}$$

<sup>5</sup> В английском языке такие отчеты обозначаются словом Hansard в честь Уильяма Хансарда (William Hansard), который впервые опубликовал британские парламентские отчеты в 1811 году.

<sup>6</sup> В данном разделе речь идет о задаче перевода с английского языка на французский. Страйтесь избегать путаницы, связанной с тем фактом, что правило Байеса требует от нас, чтобы мы рассматривали вероятность  $P(E|F)$ , а не  $P(F|E)$ , в результате чего создается впечатление, как будто перевод осуществляется с французского на английский.

Это правило указывает, что мы должны рассмотреть все возможные французские предложения  $F$  и выбрать из них то, которое максимизирует произведение  $P(E|F)P(F)$ . Коэффициент  $P(E)$  можно проигнорировать, поскольку он является одинаковым для любого  $F$ . Коэффициент  $P(F)$  представляет собой **языковую модель** для французского языка; он указывает, насколько велика вероятность появления данного конкретного предложения во французском тексте. Вероятность  $P(E|F)$  представляет собой **модель перевода**; она указывает, насколько велика вероятность того, что некоторое английское предложение будет использоваться в качестве перевода, если дано определенное французское предложение.

Внимательного читателя, безусловно, заинтересует вопрос о том, чего мы добьемся, определив вероятность  $P(F|E)$  в терминах  $P(E|F)$ . В других областях применения правила Байеса такая перестановка термов в выражениях для условной вероятности была сделана в связи с тем, что мы стремились перейти к использованию причинной модели. Например, для вычисления вероятности наличия определенных симптомов при определенном заболевании,  $P(\text{Disease}|\text{Symptoms})$ , применялась причинная модель  $P(\text{Symptoms}|\text{Disease})$ . В отличие от этого при переводе с одного языка на другой ни одно из направлений перевода не характеризуется большей причинной зависимостью, чем другое. В данном случае правило Байеса применяется в связи с тем, что мы, по-видимому, сможем легко определить с помощью обучения языковую модель  $P(F)$ , которая является более точной по сравнению с моделью перевода  $P(E|F)$  (а также более точной по сравнению с непосредственно полученной оценкой  $P(F|E)$ ). По сути такой подход позволяет разделить задачу на две части — вначале применить модель перевода  $P(F|E)$  для поиска подходящих французских предложений, в которых упоминаются те же понятия, что и в английском предложении (но это не обязательно должны быть французские предложения, полностью адекватные английскому предложению); затем воспользоваться языковой моделью  $P(F)$  (для которой имеются намного лучшие оценки вероятностей), чтобы выбрать наиболее подходящий вариант перевода.

В качестве **языковой модели**  $P(F)$  может использоваться любая модель, позволяющая присвоить предложению определенное значение вероятности. При наличии очень большой совокупности текстов можно оценить  $P(F)$  непосредственно путем подсчета количества случаев появления каждого предложения в этой совокупности текстов. Например, если с помощью Web будет собрано 100 миллионов французских предложений и обнаружено, что предложение “*Clique ici*” (Щелкните здесь) появляется 50 тысяч раз, то  $P(\text{"Clique ici"})$  равно 0,0005. Но даже при наличии 100 миллионов примеров количество экземпляров большинства возможных предложений будет равно нулю<sup>7</sup>. Поэтому мы будем использовать знакомую языковую модель двухсловных сочетаний, в которой вероятность французского предложения, состоящего из слов  $f_1 \dots f_n$ , может быть представлена следующим образом:

---

<sup>7</sup> Даже если в словаре имеется только 100 тысяч слов, то 99,99999% возможных предложений, состоящих из трех слов, будут присутствовать в совокупности текстов из 100 миллионов предложений в количестве, равном нулю. По мере увеличения длины предложений ситуация становится еще хуже.

$$P(f_1 \dots f_n) = \prod_{i=1}^n P(f_i | f_{i-1})$$

Для этого необходимо знать вероятности двухсловных сочетаний, такие как  $P("Eiffel" | "tour") = .02$ . Эти данные позволяют учитывать только самые локальные проявления синтаксических связей, в которых слово зависит лишь от предыдущего слова. Но этого часто достаточно для грубого перевода<sup>8</sup>.

Задача выбора **модели перевода**,  $P(E|F)$ , является более сложной. С одной стороны, отсутствует готовая коллекция пар предложений (английский, французский), с помощью которой можно было бы проводить обучение. С другой стороны, такая модель сложнее, поскольку в ней рассматривается перекрестное произведение предложений, а не просто отдельные предложения. Начнем с одной чрезмерно упрощенной модели перевода и постепенно усовершенствуем ее до такого уровня, чтобы она напоминала известную разработку IBM Model 3 [196], которая все еще может показаться чрезмерно упрощенной, но обнаружила свою способность вырабатывать приемлемые варианты перевода примерно в половине случаев.

Рассматриваемая чрезмерно упрощенная модель перевода основана на таком принципе: “Чтобы перевести предложение, просто переведите каждое слово отдельно, независимо от другого, в порядке слева направо”. Это — модель выбора однословного сочетания. Она позволяет легко вычислить вероятность перевода:

$$P(E|F) = \prod_{i=1}^n P(E_i | F_i)$$

В некоторых случаях эта модель действует безукоризненно. Например, рассмотрим следующую конструкцию:

$$P("the dog" | "le chien") = P("the" | "le") \times P("dog" | "chien")$$

При любом обоснованном подборе вариантов значений вероятностей выражение “the dog” (собака) будет служить наиболее правдоподобным переводом выражения “le chien”. Но в большинстве случаев прямолинейные попытки применения этой модели оканчиваются неудачей. Одна из проблем связана с порядком слов. Английское слово “dog” соответствует французскому слову “chien”, а понятие, обозначаемое в английском языке словом “brown” (коричневый), во французском языке обозначается словом “brun”. Однако словосочетание “brown dog” переводится как “chien brun”. Еще одна проблема состоит в том, что словесные обороты не связаны друг с другом в форме взаимно однозначного соответствия. Английское слово “home” часто переводят с помощью выражения “à la maison”, поэтому имеет место соответствие “один к трем” (или три к одному, при противоположном направлении перевода). Невзирая на наличие указанных проблем, разработчики модели IBM Model 3 приняли за основу жесткий

<sup>8</sup> Если в переводе нужно передать более тонкие нюансы, то модель, основанная на  $P(f_i | f_{i-1})$ , безусловно, становится неприемлемой. В качестве одного из известных примеров можно указать, что знаменитый цикл романов “A la recherche du temps perdu” (В поисках утраченного времени) Марселя Пруста объемом в 3500 страниц начинается и оканчивается одним и тем же словом, поэтому некоторые переводчики решили сделать то же самое и построили весь перевод на одном слове, находящемся примерно за 2 миллиона слов от него.

подход, по сути базирующийся на модели однословных сочетаний, но ввел несолько дополнений для компенсации ее недостатков.

Для того чтобы можно было учесть тот факт, что некоторые слова не допускают перевода один к одному, в эту модель было введено понятие **фертильности** (fertility — плодовитость) слова. Слово с фертильностью  $n$  копируется  $n$  раз, после чего каждая из этих  $n$  копий переводится независимо. Модель содержит параметры, которые показывают значение  $P(Fertility=n|word)$  для каждого французского слова. Для перевода выражения “à la maison” как выражения “home” в этой модели необходимо выбрать фертильность 0 для “à” и “la” и фертильность 1 для “maison”, а затем применить модель перевода однословных сочетаний, чтобы перевести “maison” как “home”. Такой подход кажется достаточно приемлемым, поскольку “à” и “la”, будучи словами с низким информационным содержанием, могут быть на полном основании заменены в процессе перевода пустой строкой. Но применение такого метода для перевода в другом направлении становится более сомнительным. Слову “home” должна быть назначена фертильность 3, что приведет к его преобразованию в “home home home”. Тогда первое слово “home” должно быть переведено как “à”, второе как “la” и третье как “maison”. Но с точки зрения этой модели перевода выражение “à la maison” должно иметь точно такую же вероятность, как “maison la à” (в этом и состоит та часть данного подхода, которая может быть поставлена под сомнение.) Дело в том, что выбор того или иного варианта должен осуществляться на уровне языковой модели. Может показаться, что было бы более целесообразным применение непосредственного перевода слова “home” как выражения “à la maison” вместо использования косвенного варианта с преобразованием в “home home home”, но для этого потребовалось бы больше параметров и их было бы труднее получить из доступной совокупности текстов.

В последней части этой модели перевода предусмотрена перестановка слов в правильные позиции. Такая перестановка осуществляется с помощью модели смещений, в которой указано, как следует перемещать слова из их первоначальных позиций в окончательные позиции. Например, при переводе “chien brun” как “brown dog” слово “brown” получает смещение +1 (оно сдвигается на одну позицию вправо), а слово “dog” получает смещение -1. На первый взгляд может показаться, что смещение должно быть зависимым от слова, например, такие прилагательные, как “brown”, как правило, должны иметь положительное смещение, поскольку во французском языке прилагательные обычно стоят после существительных. Но разработчики модели IBM Model 3 решили, что для реализации подхода, в котором смещения зависят от слова, потребуется слишком много параметров, поэтому смещение должно быть независимым от слова и зависимым только от положения внутри предложения, а также от длины предложений на обоих языках. Это означает, что в этой модели осуществляется оценка следующих параметров:

$$P(Offset=o | Position=p, EngLen=m, FrLen=n)$$

Таким образом, для определения смещения слова “brown” в выражении “brown dog” с помощью базы данных определяется значение  $P(Offset=1, 2, 2)$ , что может, например, привести к получению значения +1 с вероятностью 0,3 и 0 с вероятностью 0,7. Но такая модель смещений кажется еще более сомнительной, особенно тем, кто, например, пытался составить надпись из букв с магнитами на своем холодильнике и понял, что это намного сложнее, чем высказать то же самое с помощью

обычной речи. Вскоре будет показано, что такое решение было принято разработчиками не потому, что оно основано на качественной модели языка, а в связи с тем, что обеспечивает эффективное использование имеющихся данных. Так или иначе модель смещения наглядно показывает, что модель перевода среднего качества может быть значительно улучшена с помощью высококачественной языковой модели для французского языка. Ниже приведен пример, показывающий все этапы перевода одного предложения.

Исходный французский текст:	Le chien brun n' est pas allé à la maison
Модель фертильности:	1 1 1 1 1 0 1 0 0 1
Трансформированный	
французский текст:	Le chien brun n' est allé maison
Модель выбора слов:	The dog brown not did go home
Модель смещений:	0 +1 -1 +1 -1 0 0 0
Целевой английский текст:	The brown dog did not go home

Теперь нам известно, как рассчитать вероятность  $P(F|E)$  для любой пары предложений (французский, английский). Но в действительности перед нами стоит задача, получив некоторое английское предложение, найти французское предложение, которое максимизирует эту вероятность. Для этого недостаточно просто перебирать предложения, поскольку если предположить, что количество слов во французском языке равно  $10^5$ , то существует  $10^{5n}$  предложений длины  $n$ , а также много вариантов каждого из этих предложений. И даже если будут рассматриваться только 10 наиболее часто встречающихся вариантов дословного перевода для каждого слова и учитываться лишь смещения 0 или  $\pm 1$ , все равно будет получено около  $2^{n/2}10^n$  предложений, а это означает, что может быть выполнен их полный перевод при  $n=5$ , но не при  $n=10$ . Поэтому вместо перебора необходимо осуществлять поиск наилучшего решения. Практика показала, что эффективным является поиск на основе алгоритма A\*; см. [545].

### Определение с помощью обучения вероятностей для машинного перевода

Выше была кратко описана модель для  $P(F|E)$ , которая предусматривает применение четырех перечисленных ниже множеств параметров.

- Языковая модель.  $P(\text{word}_i | \text{word}_{i-1})$ .
- Модель фертильности.  $P(\text{Fertility}=n | \text{word}_F)$ .
- Модель выбора слова.  $P(\text{word}_E | \text{word}_F)$ .
- Модель смещения.  $P(\text{Offset}=o | \text{pos}, \text{len}_E, \text{len}_F)$ .

Но даже при использовании скромного словаря, состоящего из 1000 слов, для этой модели требуются миллионы параметров. Очевидно, что необходимо обеспечить определение этих параметров с помощью обучения на основе данных. Предположим, что единственными доступными данными является двуязычная совокупность текстов. Ниже описан способ использования этих данных.

- Сегментация на предложения. Единицей перевода является предложение, поэтому нам потребуется разбить совокупность текстов на предложения. Надежным показателем конца предложения является точка, но в таком фрагменте

текста, как “Dr. J. R. Smith of Rodeo Dr. arrived.”, признаком конца предложения является только последняя точка. Сегментация на предложения может быть выполнена с точностью около 98%.

- Оценка языковой модели для французского языка  $P(\text{word}_i | \text{word}_{i-1})$ . Рассматривая только французскую половину совокупности текстов, подсчитать частоты пар слов и выполнить выравнивание, чтобы получить оценку  $P(\text{word}_i | \text{word}_{i-1})$ . Например, может быть получено значение  $P("Eiffel" | "tour") = .02$ .
- Выравнивание предложений. Для каждого предложения в английской версии определить, какое предложение (предложения) соответствует ему во французской версии. Обычно следующее предложение в английском тексте соответствует следующему предложению во французском тексте в форме согласования “один к одному”, но иногда возникают другие варианты: одно предложение на одном из языков может быть разбито на два, что приводит к согласованию “два к одному”, или может быть изменен на противоположный порядок следования двух предложений, а это приведет к согласованию “два к двум”. Выравнивание предложений (“один к одному”, “один к двум” или “два к двум” и т.д.) может быть обеспечено только на основании сравнения длины предложений с точностью в пределах от 90 до 99% с использованием одного из вариантов алгоритма сегментации Виттерби (см. листинг 23.1). С применением отметок, общих для обоих языков, таких как числа или имена собственные, а также слов, которые, как известно, имеют в двуязычном словаре однозначный перевод, можно добиться еще лучшего выравнивания.

Теперь можно приступить к оценке параметров модели перевода. Такую задачу можно решить, приняв довольно слабое начальное предположение, а затем постепенно его улучшая, как описано ниже.

- Оценка начальной модели фертильности  $P(\text{Fertility}=n | \text{word}_F)$ . Найдя французское предложение длины  $m$ , которое выравнивается с английским предложением длины  $n$ , будем рассматривать его как свидетельство того, что каждое французское слово имеет фертильность  $n/m$ . Рассмотрим все свидетельства во всех предложениях, чтобы получить распределение вероятностей фертильности для каждого слова.
- Оценка начальной модели выбора слова  $P(\text{word}_E | \text{word}_F)$ . Рассмотрим все французские предложения, которые содержат, скажем, слова “brun”. Слова, которые появляются наиболее часто в английских предложениях, выравниваемых с этими предложениями, являются наиболее вероятными буквальными переводами слова “brun”.
- Оценка начальной модели смещения  $P(\text{Offset}=\delta | pos, len_E, len_F)$ . Теперь, после получения модели выбора слова, воспользуемся ею, чтобы получить оценку модели смещения. Для каждого английского предложения длины  $n$ , которая выравнивается с французским предложением длины  $m$ , проанализировать каждое французское слово в предложении (в позиции  $i$ ) и каждое английское слово в предложении (в позиции  $j$ ), которое является наиболее вероятным вариантом выбора для французского слова, и рассматривать его как свидетельство для вероятности  $P(\text{Offset}=\delta | i, n, m)$ .

- Усовершенствование всех оценок. Воспользоваться алгоритмом EM (expectation–maximization — ожидание–максимизация), чтобы усовершенствовать оценки. Скрытой переменной является **вектор выравнивания слов** между парами предложений, выровненными по предложениям. Этот вектор указывает для каждого английского слова позицию соответствующего французского слова во французском предложении. Например, может быть получено следующее:

Исходный французский текст: Le chien brun n' est pas allé à la maison  
 Целевой английский текст: The brown dog did not go home  
 Вектор выравнивания слов: 1 3 2 5 4 7 10

Вначале с использованием текущих оценок параметров создадим вектор выравнивания слов для каждой пары предложений. Это позволит нам получать лучшие оценки. Модель фертильности оценивается путем подсчета того, сколько раз один из элементов вектора выравнивания слов указывает на несколько слов или на нулевое количество слов. После этого в модели выбора слов могут рассматриваться только те слова, которые выровнены друг с другом, а не все слова в предложениях, тогда как в модели смещений может рассматриваться каждая позиция в предложении для определения того, насколько часто она смещается в соответствии с вектором выравнивания слов. К сожалению, точно не известно, каковым является правильное выравнивание, а количество вариантов выравнивания слишком велико для того, чтобы перебрать их все. Поэтому мы вынуждены осуществлять поиск выравниваний с высокой вероятностью и взвешивать их по их вероятностям, собирая свидетельства для новых оценок параметров. Это все, что требуется для алгоритма EM. На основании начальных параметров вычисляются выравнивания, а с помощью выравниваний уточняются оценки параметров. Такая процедура повторяется до полной сходимости.

## 23.5. РЕЗЮМЕ

Основные положения, изложенные в этой главе, перечислены ниже.

- Вероятностные языковые модели, основанные на  $n$ -элементных сочетаниях, позволяют получить весьма значительный объем информации о языке.
- Контекстно-свободные грамматики (Context-Free Grammar — CFG) могут быть расширены до вероятностных контекстно-свободных грамматик, которые позволяют определять их параметры с помощью обучения из имеющихся данных, а также легче решать задачу устранения неоднозначности.
- В системах **информационного поиска** используется очень простая языковая модель, основанная на обработке мультимножеств слов, но даже эта модель позволяет достичь высоких показателей **полноты и точности** на очень больших совокупностях текстов.
- В системах **извлечения информации** используется более сложная модель, которая включает простейшие синтаксические и семантические конструкции. Для реализации таких систем часто применяются каскады конечных автоматов.
- В практически применимых системах **машинного перевода** используется целый ряд методов, начиная от полного синтаксического и семантического анализа и заканчивая статистическими методами, основанными на учете частот слов.

- При формировании статистической языковой системы лучше всего опереться на модель, позволяющую эффективно использовать имеющиеся данные, даже если эта модель кажется чрезмерно упрощенной.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Подход с применением моделей  $n$ -буквенных сочетаний для моделирования языка был предложен Марковым [983]. Клод Шенон [1394] впервые создал модели  $n$ -словных сочетаний английского языка. Хомский [250], [251] указал на ограничения моделей на основе конечных автоматов по сравнению с моделями на основе контекстно-свободных грамматик и пришел к заключению: “Вероятностные модели не позволяют каким-то образом добиться лучшего понимания некоторых основных проблем синтаксической структуры”. Это утверждение справедливо, но в нем игнорируется тот факт, что вероятностные модели обеспечивают лучшее понимание некоторых других основных проблем, а именно тех проблем, которые не могут быть решены с помощью контекстно-свободных грамматик. Замечания, сделанные Хомским, оказали неблагоприятный эффект, выразившийся в том, что в течение двух десятилетий многие исследователи избегали использования статистических моделей. Положение изменилось лишь после того, как указанные модели снова вышли на передний план и стали применяться при распознавании речи [730].

Метод сглаживания с добавлением единицы был предложен Джейфри [728], а метод сглаживания с удалением путем интерполяции разработан Елинеком и Мерсером [732], которые использовали этот метод для распознавания речи. В число других методов входят сглаживание Виттена–Белла [1605] и сглаживание Гуда–Тьюринга [257]. Последний метод также широко применяется при решении задач биоинформатики. Проблематика биостатистических и вероятностных задач NLP постепенно сближается, поскольку в каждой из этих областей приходится иметь дело с длинными структурированными последовательностями, выбранными из алфавита непосредственных составляющих.

Простые модели  $n$ -буквенных и  $n$ -словных сочетаний не являются единственными возможными вероятностными моделями. В [136] описана вероятностная модель текста, называемая **скрытым распределением Дирихле**, в которой документ рассматривается как комбинация тем, а каждая из тем характеризуется собственным распределением слов. Эта модель может рассматриваться как дополнение и уточнение модели **скрытой семантической индексации** Дирвестера [376] (см. также [1169]); кроме того, она тесно связана с моделью сочетания многочисленных причин [1345].

В **вероятностных контекстно-свободных грамматиках** (Probabilistic Context-Free Grammar — PCFG) устранены все недостатки вероятностных моделей, отмеченные Хомским, и они показали свои преимущества над обычными контекстно-свободными грамматиками. Грамматики PCFG были исследованы Бутом [151] и Саломаа [1346]. В [729] представлен алгоритм декодирования стека, представляющий собой один из вариантов алгоритма поиска Виттерби, который может использоваться для определения наиболее вероятной версии синтаксического анализа с помощью грамматики PCFG. В [63] представлен внешний–внутренний алгоритм, а в [889] описаны области его применения и ограничения. В [236] и [804] обсуждаются проблемы синтаксического анализа с помощью грамматик в виде **банка деревьев**.

В [1467] показано, как определять с помощью обучения грамматические правила на основе слияния байесовских моделей. Другие алгоритмы для грамматик PCFG представлены в [235] и [980]. В [282] приведен обзор результатов, полученных в этой области, и даны пояснения к одной из наиболее успешных программ статистического синтаксического анализа.

К сожалению, грамматики PCFG при выполнении самых различных задач показывают более низкую производительность по сравнению с простыми  $n$ -элементными моделями, поскольку грамматики PCFG не позволяют представить информацию, связанную с отдельными словами. Для устранения этого недостатка некоторые авторы [281], [237], [713] предложили варианты **лексикализованных вероятностных грамматик**, в которых совместно используются контекстно-свободные грамматики и статистические данные, касающиеся отдельных слов.

Первой попыткой собрать сбалансированную совокупность текстов для эмпирической лингвистики явилось создание коллекции Brown Corpus [493]. Эта совокупность состояла примерно из миллиона слов с отметками, обозначающими части речи. Первоначально эта коллекция хранилась на 100 тысячах перфокарт. Банк деревьев синтаксического анализа Пенна [982] представляет собой коллекцию, состоящую примерно из 1,6 миллиона слов текста, для которого вручную выполнен синтаксический анализ с преобразованием в деревья. Эта коллекция помещается на компакт-диске. В издании British National Corpus [905] данная коллекция была расширена до 100 миллионов слов. В World Wide Web хранится свыше триллиона слов больше чем на 10 миллионах серверов.

В последнее время растет интерес к области **информационного поиска**, обусловленный широким применением поиска в Internet. В [1296] приведен обзор ранних работ в этой области и представлен принцип ранжирования вероятностей. В [980] дано краткое введение в проблематику информационного поиска в контексте статистических подходов к решению задач NLP. В [59] приведен обзор общего назначения, заменивший более старые классические работы [492] и [1347]. Книга *Managing Gigabytes* [1606] посвящена решению именно той задачи, о которой говорит ее название, — описанию того, как можно эффективно индексировать, применять сжатие и выполнять запросы применительно к совокупности текстов гигабайтовых размеров. В рамках конференции TREC, организованной Национальным институтом стандартов и технологии (National Institute of Standards and Technology — NIST) при правительстве Соединенных Штатов, проводятся ежегодные соревнования между системами информационного поиска и публикуются труды с описанием достигнутых результатов. За первые семь лет таких соревнований производительность участников в них программ выросла примерно в два раза.

Наиболее широко применяемой моделью для информационного поиска является **модель векторного пространства** Салтона [1348]. В первые годы развития этой области указанная работа Салтона была фактически самой влиятельной. Имеются также две альтернативные вероятностные модели. Модель, представленная в этой книге, основана на [1225]. В ней моделируется совместное распределение вероятностей  $P(D, Q)$  в терминах  $P(Q|D)$ . В другой модели [985], [1297] используется вероятность  $P(D|Q)$ . В [879] показано, что обе эти модели основаны на одном и том же совместном распределении вероятностей, но от выбора модели зависит то, какие методы должны применяться для определения параметров с помощью обучения. Описание,

приведенное в данной главе, основано на обеих этих моделях. В [1522] приведено сравнение различных моделей информационного поиска.

В [187] описана реализация машины поиска для World Wide Web, включая алгоритм PageRank, в основе которого лежит независимый от запроса критерий качества документа, базирующийся на анализе Web-ссылок. В [805] описано, как находить авторитетные источники информации в Web с использованием анализа ссылок. В [1411] приведены результаты исследования журнала с данными о миллиарде поисковых операций, выполненных в Web. В [864] приведен обзор литературы по исправлению орфографических ошибок. В [1230] описан классический алгоритм выделения основы с помощью правил, а в [860] описан вариант, в котором применяется словарь.

В [980] приведен хороший обзор проблематики классификации и кластеризации документов. В [738] используются теория статистического обучения и теория машин векторов поддержки для теоретического анализа ситуаций, в которых классификация должна быть успешной. В [37] приведены данные о том, что при классификации новостных сообщений агентства Reuters, относящихся к категории “Earnings” (Доходы), была достигнута точность 96%. В [824] приведены данные о том, что при использовании наивного байесовского классификатора достигается точность вплоть до 95%, а при использовании байесовского классификатора, в котором учитываются некоторые зависимости между характеристиками, — вплоть до 98,6%. В [922] приведен обзор результатов, достигнутых за сорок лет применения наивных байесовских моделей для классификации и поиска в тексте.

Последние достижения в этой области публикуются в журнале *Information Retrieval* и в трудах ежегодной конференции *SIGIR*.

Одними из первых программ извлечения информации являются Gus [143] и Frump [380]. В основе некоторых проектов современных систем извлечения информации лежат работы в области семантических грамматик, проводившиеся в 1970-х и 1980-х годах. Например, в интерфейсе системы резервирования авиабилетов с семантической грамматикой используются такие категории, как *Location* (место нахождения) и *FlyTo* (место назначения), а не *NP* и *VP*. Описание результатов реализации одной из систем, основанных на семантических грамматиках, приведено в [130].

Новейшие результаты исследований по извлечению информации пропагандируются на ежегодных конференциях MUC (Message Understanding Conference), спонсором которых выступает правительство США. Система FASTUS была разработана Хоббсом и др. [664]; в сборнике статей, в котором впервые была опубликована информация об этой системе [1299], можно найти информацию и о других системах, в которых используются модели конечных автоматов.

В 1930-м году Петр Троянский (Petr Troyanskii) подал заявку на патент, в котором была сформулирована идея “машины перевода”, но в то время еще не существовали компьютеры, позволяющие реализовать эту идею. В марте 1947 года Уоррен Вивер (Warren Weaver), сотрудник Фонда Рокфеллера, написал Норберту Винеру письмо, в котором указал, что решение задачи машинного перевода вполне возможно. Опираясь на работы в области криптографии и теории информации, Вивер писал: “Когда я рассматриваю статью, написанную на русском языке, я говорю себе: «Она фактически написана на английском языке, но закодирована странными символами. Теперь я приступаю к ее декодированию»”. В течение следующего десятилетия все сообщество специалистов в этой области предпринимало упорные попытки декодирования текстов на иностранном языке таким способом. Компания IBM про-

демонстрировала соответствующую зачаточную систему в 1954 году. Энтузиазм, характерный для этого периода, показан в [69] и [942]. Последующее разочарование в возможностях машинного перевода описано Линдсеем [935], указавшим также на некоторые препятствия, связанные с необходимостью обеспечения взаимодействия синтаксиса и семантики, а также с потребностью в наличии знаний о мире, с которыми сталкивается машинный перевод. Правительство США выразило недовольство полным отсутствием прогресса в этой области и сформулировало свое заключение в одном из отчетов, который известен как отчет ALPAC [21]: “Нет ни ближайших, ни обозримых перспектив создания практически применимых систем машинного перевода”. Однако работы в ограниченном объеме продолжались, и в BBC США в 1970 году была развернута система Systran, которая была взята на вооружение Европейским экономическим сообществом в 1976 году. В том же 1976 году была развернута система перевода сообщений о погоде Taut-Meteo [1255]. К началу 1980-х годов возможности компьютеров возросли до такой степени, что выводы отчета ALPAC потеряли свою актуальность. В [1548] приведены сведения о некоторых новейших приложениях машинного перевода, основанных на системе Wordnet. Учебное введение в эту область приведено в [710].

Первые предложения по использованию статистического машинного перевода были сделаны в заметках Уоррена Вивера, опубликованных в 1947 году, но возможность практического применения этих методов появилась только в 1980-х годах. Описание этой тематики, приведенное в данной главе, основано на работе Брауна и его коллег из компании IBM [195], [196]. Эти труды весьма насыщены математической символикой, поэтому прилагаемый к ним учебник Кевина Найта [806] воспринимается как глоток свежего воздуха. В более современных исследованиях по статистическому машинному переводу наблюдается отказ от модели двухсловных сочетаний в пользу моделей, которые включают некоторые синтаксические конструкции [1627]. Первые работы в области сегментации предложений были выполнены Палмером и Херстом [1166]. Задача выравнивания двуязычных предложений рассматривается в [1042].

Есть две превосходные книги по вероятностной обработке лингвистической информации: книга [235] является краткой и точной, а книга [980] — всеобъемлющей и современной. С состоянием работ по созданию практических методов обработки лингвистической информации можно ознакомиться по материалам проводимой один раз в два года конференции *Applied Natural Language Processing* (ANLP) и конференции *Empirical Methods in Natural Language Processing* (EMNLP), а также по публикациям в журнале *Natural Language Engineering*. Организация SIGIR финансирует выпуск информационного бюллетеня и проведение ежегодной конференции по информационному поиску.

## УПРАЖНЕНИЯ

---

- 23.1.**  (*Adaptировано из [756].*) В этом упражнении предлагается разработать классификатор для выявления авторства: при наличии некоторого текста этот классификатор должен попытаться определить, какой из двух возможных авторов написал этот текст. Получите образцы текста двух различных авторов. Разделите их на обучающие и контрольные множества. После этого определи-

те с помощью обучения параметры модели однословных сочетаний для каждого автора по обучающему множеству. Наконец, для каждого контрольного множества рассчитайте его вероятность в соответствии с каждой моделью однословных сочетаний и присвойте эту вероятность наиболее вероятной модели. Оцените точность этого метода. Можете ли вы повысить его точность с помощью дополнительных характеристик? Эта подобласть лингвистики называется **стилометрией**; к числу достижений в этой области относится идентификация автора “Заметок федералиста” (Federalist Papers) [1091] и некоторых произведений Шекспира, авторская принадлежность которых некогда спорилась [486].

- 23.2. В этом упражнении исследуется качество моделей  $n$ -элементных сочетаний, характерных для некоторого языка. Найдите или создайте мноязыковую совокупность, состоящую примерно из 100 тысяч слов. Сегментируйте ее на слова и вычислите частоту каждого слова. Каково количество присутствующих в ней различных слов? Начертите график зависимости частоты слов от их ранга (первое, второе, третье...) с логарифмической шкалой по горизонтали и по вертикали. Кроме того, подсчитайте частоты двухсловных сочетаний (два подряд идущих слова) и трехсловных сочетаний (три подряд идущих слова). Воспользуйтесь этими частотами для генерации языка: на основании моделей одно-, двух- и трехсловных сочетаний последовательно сформируйте образцы текста из 100 слов, выполняя выбор случайным образом в соответствии со значениями частот. Сравните три сформированных текста с фактически имеющимся текстом на рассматриваемом языке. Наконец, рассчитайте показатель связности каждой модели.
- 23.3. В этом упражнении рассматривается задача распознавания нежелательной электронной почты (спама). *Спамом* принято называть незатребованные объемистые коммерческие сообщения, поступающие по электронной почте. Утомительную задачу разборки спама приходится решать многим пользователям, поэтому создание надежного способа его устраниния явилось бы большим достижением. Создайте две совокупности текстов — состоящую из почтовых сообщений, представляющих собой спам, и состоящую из обычных почтовых сообщений. Исследуйте каждую совокупность и определите, какие характеристики, скорее всего, окажутся применимыми для классификации: однословные сочетания, двухсловные сочетания, длина сообщений, отправитель, время получения и т.д. Затем проведите обучение алгоритма классификации (дерева решений, наивной байесовской модели или какого-то другого выбранного вами алгоритма) на обучающем множестве и определите его точность на контрольном множестве.
- 23.4. Создайте контрольное множество из пяти запросов и предъявите эти запросы трем основным машинам поиска Web. Оцените каждую из них по показателю точности для 1, 3 и 10 возвращенных документов и по среднему обратному рангу. Попытайтесь объяснить обнаруженные различия.
- 23.5. Попытайтесь определить, в какой из машин поиска, рассматриваемых в предыдущем упражнении, используются методы приведения к нижнему регистру, выделения основы, выявления синонимов и исправления орфографических ошибок.

- 23.6. Оцените, какой объем памяти является необходимым для индекса к совокупности Web-страниц, состоящей из миллиарда страниц. Укажите, какие предположения были вами приняты.
- 23.7. Напишите регулярное выражение или короткую программу для извлечения названий компаний. Проверьте ее на совокупности, состоящей из деловых новостных сообщений. Определите полноту и точность полученных результатов.
- 23.8. Выберите пять предложений и передайте их в оперативную службу перевода. Переведите их с английского на другой язык, а затем снова на английский. Оцените, насколько полученные при этом предложения являются грамматически правильными и сохранившими смысл. Повторите этот процесс; будут ли во второй итерации получены худшие результаты или такие же результаты? Влияет ли на качество результатов выбор промежуточного языка?
- 23.9. Соберите некоторые примеры выражений с указанием времени, таких как “two o'clock”, “midnight” и “12:46”. Кроме того, подготовьте некоторые примеры, являющиеся грамматически неправильными, такие как “thirteen o'clock” или “half past two fifteen”. Напишите грамматику для языка выражений с указанием времени.
- 23.10. (*Адаптировано из [806].*) В модели машинного перевода IBM Model 3 предполагается, что после того как с помощью модели выбора слов будет подготовлен список слов, а с помощью модели смещения будут подготовлены возможные перестановки слов, можно будет применить языковую модель для выбора наилучшей перестановки. Данное упражнение посвящено исследованию того, насколько обоснованным является указанное предположение. Попытайтесь переставить слова в приведенных ниже предложениях, подготовленных с помощью модели IBM Model 3, в правильном порядке.
- have programming a seen never I language better
  - loves john mary
  - is the communication exchange of intentional information brought by about the production perception of and signs from drawn a of system signs conventional shared

С какими предложениями вам удалось справиться? Знания какого типа пришлось вам для этого привлечь? Проведите обучение модели двухсловных сочетаний с помощью обучающей совокупности и воспользуйтесь этой моделью для поиска перестановок некоторых предложений из контрольной совокупности с наибольшей вероятностью. Определите точность этой модели.

- 23.11. Согласно данным англо-французского словаря, переводом для слова “hear” является глагол “entendre”. Но если проводится обучение модели IBM Model 3 по отчетам канадского парламента, то наиболее вероятным переводом для слова “hear” становится “Bravo”. Объясните, почему такое происходит, и оцените, каким может быть распределение фертильности для слова “hear”. (*Подсказка.* Вам может потребоваться ознакомиться с каким-то текстом парламентского отчета. Попробуйте выполнить поиск в Web с помощью запроса [Hansard hear].)

# 24 ВОСПРИЯТИЕ

*В данной главе речь идет о том, как ввести в компьютер исходные, необработанные данные, полученные из реального мира.*

❖ **Восприятие** предоставляет агентам информацию о мире, в котором они обитают. Восприятие осуществляется с помощью ❖ **датчиков**. Датчиком может быть любое устройство, позволяющее зафиксировать состояние какого-то аспекта среды и передать полученные данные в качестве входных в программу агента. Датчик может быть настолько простым, как однобитовый детектор, который лишь определяет, разомкнут или замкнут выключатель, или настолько сложным, как сетчатка человеческого глаза, которая содержит больше ста миллионов фоточувствительных элементов. В данной главе наше внимание будет сосредоточено на зрении, поскольку оно намного превосходит по информативности все остальные чувства, когда приходится сталкиваться с проявлениями физического мира.

## 24.1. ВВЕДЕНИЕ

В распоряжении искусственных агентов имеется целый ряд *сенсорных модальностей*. К числу тех из них, которые являются общими и для людей, и для роботов, относятся зрение, слух и осязание. Проблема слухового восприятия, по крайней мере, в той части, которая касается восприятия речи, рассматривалась в разделе 15.6. Осязание, или **тактильное восприятие**, будет рассматриваться в главе 25, где речь идет о его использовании в сложнейших манипуляциях, выполняемых роботами, а остальная часть настоящей главы будет посвящена зрению. Некоторые роботы способны воспринимать модальности, не доступные людям, не пользующимся специальными приспособлениями, такие как радиоволны, инфракрасные лучи, сигналы глобальной системы навигации и определения положения (Global Positioning System — GPS) и другие беспроводные сигналы. Некоторые роботы осуществляют **активное восприятие**; это означает, что они посыпают такие импульсные сигналы, как радарные или ультразвуковые, и принимают отражение этих импульсов от среды.

Существуют два способа, с помощью которых агент может использовать полученные им результаты восприятия. В подходе, основанном на **извлечении характеристик**, агенты распознают какое-то небольшое количество характеристик в полученных ими сенсорных входных данных и передают эти данные непосредственно в свою

программу агента, которая может вырабатывать команды по осуществлению действий, представляющих собой реакцию на изменение этих характеристик, или применять эти данные в сочетании с другой информацией. По такому принципу действовал агент в мире вампира, оборудованный пятью датчиками, каждый из которых извлекал информацию об одной однобитовой характеристике. Кроме того, недавно стало известно, что в нервной системе мухи извлекаются данные о характеристиках из оптического потока и эти данные направляются прямо на мускулы, которые помогают мухе управлять своим движением в воздухе, что дает ей возможность быстро реагировать и изменять направление полета в течение 30 миллисекунд.

Альтернативным этому подходу является подход **на основе модели**, в котором сенсорные стимулы используются для реконструкции модели мира. При этом подходе работа начинается с функции  $f$ , которая отображает состояние мира  $W$  на стимулы  $S$ , создаваемые этим миром:

$$S = f(W)$$

Функция  $f$  определена в физике и в оптике, а также достаточно хорошо изучена. Задача выработки стимулов  $S$  с использованием функции  $f$  и данных о реальном или воображаемом мире  $W$  решается в области **компьютерной графики**. Задача машинного зрения в определенном смысле является обратной задаче компьютерной графики — в ней предпринимается попытка вычислить  $W$  с помощью  $f$  и  $S$  по следующей формуле:

$$W = f^{-1}(S)$$

К сожалению, функция  $f$  не имеет приемлемой обратной функции. Прежде всего, мы не можем заглянуть за угол, поэтому не имеем возможности восстановить все аспекты мира из полученных зрительных стимулов. Более того, даже наблюдаемая часть мира представляется как чрезвычайно неоднозначная — без дополнительной информации нельзя сказать, содержит ли стимул  $S$  изображение игрушечного ящера Годзилла, ломающего шестидесятиметровый макет здания, или в нем представлен настоящий монстр, разрушающий здание высотой шестьдесят метров. Некоторые из подобных проблем можно решить, составив распределение вероятностей по мирам, а не пытаясь найти уникальный мир:

$$P(W) = P(W|S) \cdot P(S)$$

Еще более важным недостатком моделирования такого типа является предпринимаемая в нем попытка решить слишком трудную проблему. Достаточно сказать, что в компьютерной графике может потребоваться несколько часов вычислений для того, чтобы прорисовать единственный кадр кинофильма, притом что в секунду требуется 24 таких кадра; к тому же вычисление функции  $f^{-1}$  гораздо сложнее по сравнению с вычислением  $f$ . Очевидно, что такой объем вычислений слишком велик даже для суперкомпьютера, не говоря уже об обычной мухе, если требуется обеспечить реагирование в реальном времени. К счастью, агенту не требуется модель с таким уровнем детализации, который используется в компьютерной графике, где нужно добиться, чтобы построенное изображение стало таким же реальным, как настоящая фотография. Агенту достаточно знать, скрывается ли в кустарнике тигр, а не учитывать все данные о точном местонахождении и ориентации каждого волоска на спине этого тигра.

Основная часть настоящей главы посвящена описанию средств, позволяющих обеспечить распознавание объектов, таких как притаившиеся тигры, и в ней будут показаны способы решения этой задачи без представления данных о самом тигре до мельчайших подробностей. В разделе 24.2 описан процесс формирования изображения и определены некоторые особенности функции  $f(W)$ . Вначале рассматривается геометрия этого процесса. Будет описано, как свет отражается от объектов во внешнем мире и попадает на точки в плоскости изображения оптического датчика искусственного агента. Геометрия объясняет, почему большой ящер Годзилла, находящийся далеко от нас, кажется таким же, как маленький ящер Годзилла, расположенный намного ближе. После этого рассматривается фотометрия данного процесса, что позволяет понять, как свет в наблюдаемой сцене определяет яркость точек на изображении. Геометрия и фотометрия, вместе взятые, позволяют получить модель того, как объекты во внешнем мире отображаются на двухмерный массив пикселов.

Получив представление о том, как формируются изображения, мы перейдем к изучению способов их обработки. Процесс обработки потока визуальной информации как людьми, так и компьютерами можно разделить на три этапа. На раннем этапе обработки, называемом *зрением низкого уровня* (раздел 24.3), необработанное изображение сглаживается для устранения шума и извлекаются характеристики двухмерного изображения, в частности данные о краях участков, разделяющих регионы изображения. На этапе визуальной обработки среднего уровня эти края группируются в целях формирования двухмерных областей. А на этапе обеспечения зрения высокого уровня (раздел 24.4) эти двухмерные области распознаются как действительные объекты в реальном мире (раздел 24.5). Мы изучим различные элементы изображения, позволяющие более успешно решать эту задачу, включая признаки движения, стереоданные, текстуру, затенение и контуры. Задача распознавания объектов важна для агентов, действующих в условиях дикой природы, чтобы они могли обнаруживать присутствие тигров, а также важна для промышленных роботов, чтобы они могли отличать гайки от болтов. Наконец, в разделе 24.6 описано, как можно использовать результаты распознавания объектов для выполнения полезных задач, таких как манипулирование объектами и навигация. Средства манипулирования позволяют захватывать и использовать инструменты и другие объекты, а средства навигации дают возможность передвигаться из одного места в другое без столкновений с какими-либо препятствиями. Не упуская из виду эти задачи, можно добиться того, чтобы агент формировал модель только в таком объеме, который позволяет ему успешно достичь своих целей.

## 24.2. ФОРМИРОВАНИЕ ИЗОБРАЖЕНИЯ

В процессе зрения концентрируется свет, рассеянный объектами в *сцене*, и создается двухмерное *изображение* на плоскости изображения. Плоскость изображения покрыта светочувствительным материалом — в сетчатке таковым являются молекулы родопсина, на фотографической пленке — галогены серебра, а в цифровой камере — массив элементов с зарядовой связью (Charge-Coupled Device — CCD). Каждый элемент в приборе с зарядовой связью (ПЗС) накапливает заряд, пропорциональный количеству электронов, освобожденных в результате поглощения фотонов за фиксированный период времени. В цифровой камере плоскость

изображения представлена в виде прямоугольной решетки, состоящей из нескольких миллионов **пикселов**. В глазу имеется аналогичный массив элементов, состоящий примерно из 100 миллионов палочек и 5 миллионов колбочек, сгруппированных в гексагональный массив.

Сцена очень велика, а плоскость изображения весьма мала, поэтому требуется определенный способ фокусировки света на плоскости изображения. Такая операция может быть выполнена с помощью линзы или без нее. В любом случае наша основная задача состоит в том, чтобы определить геометрию происходящих преобразований и обеспечить возможность прогнозировать, где каждая точка сцены найдет свое представление на плоскости изображения.

### Получение изображения без линз — камера-обскура

Простейший способ формирования изображения состоит в использовании **камеры-обскуры**, в конструкцию которой входит микроотверстие  $O$  в передней части ящика и плоскость изображения в задней части ящика (рис. 24.1). Мы будем использовать трехмерную систему координат с началом координат в точке  $O$  и рассматривать точку  $P$  в сцене, имеющую координаты  $(X, Y, Z)$ . Точка  $P$  проектируется в точку  $P'$  на плоскости изображения с координатами  $(x, y, z)$ . Если  $f$  — расстояние от микроотверстия до плоскости изображения, то с помощью теоремы подобия треугольников можно получить следующие уравнения:

$$\frac{-x}{f} = \frac{X}{Z}, \quad \frac{-y}{f} = \frac{Y}{Z} \Rightarrow x = \frac{-fX}{Z}, \quad y = \frac{-fY}{Z}$$

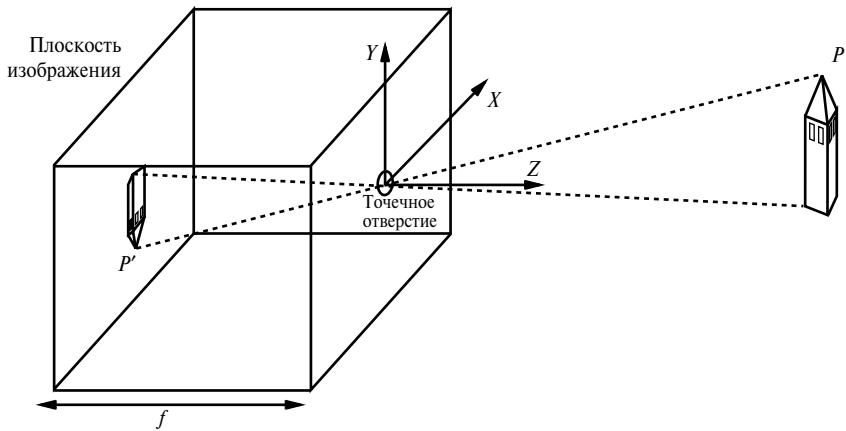


Рис. 24.1. Геометрия формирования изображения в камере-обскуре

Эти уравнения определяют процесс формирования изображения, называемый **перспективной проекцией**. Заслуживает внимания то, что значение координаты  $Z$  находится в знаменателе, а это означает, что чем дальше объект, тем меньше его изображение. Кроме того, наличие знака “минус” означает, что изображение инвертировано, т.е. повернуто на  $180^\circ$  по сравнению с самой сценой.

При перспективной проекции параллельные линии сходятся в одной точке на горизонте (достаточно представить себе уходящие вдаль железнодорожные рельсы).

Рассмотрим, почему так должно быть. Линия в сцене, проходящая через точку  $(X_0, Y_0, Z_0)$  в направлении  $(U, V, W)$ , может быть описана как множество точек  $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$ , где  $\lambda$  изменяется в пределах от  $-\infty$  до  $+\infty$ . Проекция точки  $P_\lambda$  от этой линии до плоскости изображения задается следующей формулой:

$$\left( f \frac{X_0 + \lambda U}{Z_0 + \lambda W}, f \frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right)$$

По мере того как  $\lambda \rightarrow \infty$  или  $\lambda \rightarrow -\infty$ , эта формула принимает вид  $p_\infty = (fU/W, fV/W)$ , если  $W \neq 0$ . Точку  $p_\infty$  называют **точкой схода**, связанной с семейством прямых линий с ориентацией  $(U, V, W)$ . Все линии, имеющие одну и ту же ориентацию, имеют и одинаковую точку схода.

Если объект имеет относительно небольшую глубину по сравнению с его расстоянием от камеры, появляется возможность аппроксимировать перспективную проекцию с помощью **масштабированной ортогональной проекции**. Идея такой операции состоит в следующем: если глубина  $Z$  точек объекта изменяется в некоторых пределах  $Z_0 \pm \Delta Z$ , где  $\Delta Z \gg Z_0$ , то коэффициент перспективного масштабирования  $f/Z$  можно приближенно представить с помощью константы  $s = f/Z_0$ . Уравнения для проекции, которые связывают координаты сцены  $(X, Y, Z)$  с координатами плоскости изображения, принимают вид  $x = sX$  и  $y = sY$ . Следует отметить, что масштабированная ортогональная проекция представляет собой аппроксимацию, действительную только для таких частей сцены, которые не характеризуются значительными изменениями внутренней глубины; эта проекция должна использоваться только для исследования свойств “в малом”, а не “в большом”. В качестве примера, позволяющего убедиться в необходимости соблюдать осторожность, отметим, что при использовании ортогональной проекции параллельные линии остаются параллельными, а не сливаются в точке схода!

## Системы линз

В глазах позвоночных и в современных видеокамерах используются **линзы**. Линза имеет гораздо большую площадь по сравнению с микроотверстием, что позволяет пропускать с ее помощью больше света. За это приходится платить тем, что исчезает возможность представить в резком фокусе всю сцену одновременно. Изображение объекта в сцене, находящегося на расстоянии  $Z$ , создается на фиксированном расстоянии от линзы  $Z'$ , а отношение между  $Z$  и  $Z'$  задается с помощью следующего уравнения линзы, где  $f$  — фокусное расстояние линзы:

$$\frac{1}{Z} + \frac{1}{Z'} = \frac{1}{f}$$

Если дана определенная возможность выбора расстояния изображения  $Z_0'$  между узловой точкой линзы и плоскостью изображения, то точки сцена с глубинами в диапазоне, близком к  $Z_0$ , где  $Z_0$  — соответствующее расстояние до объекта, могут быть спроектированы на изображение в достаточно резком фокусе. Указанный диапазон глубин в сцене называется **глубиной резкости пространственного изображения**.

Следует отметить, что расстояние до объекта  $Z$  обычно намного больше по сравнению с расстоянием до изображения  $Z'$  или по сравнению с  $f$ , поэтому часто можно воспользоваться следующей аппроксимацией:

$$\frac{1}{Z} + \frac{1}{Z'} \approx \frac{1}{Z'} \Rightarrow \frac{1}{Z'} \approx \frac{1}{f}$$

Таким образом, расстояние до изображения  $Z' \approx f$ . Поэтому можно по-прежнему использовать уравнения перспективной проекции камеры-обскуры для описания геометрии формирования изображения в системе линз.

Для того чтобы можно было создавать сфокусированные изображения объектов, находящихся на разных расстояниях  $Z$ , линза в глазу (рис. 24.2) меняет форму, а линза в камере передвигается в направлении  $Z$ .

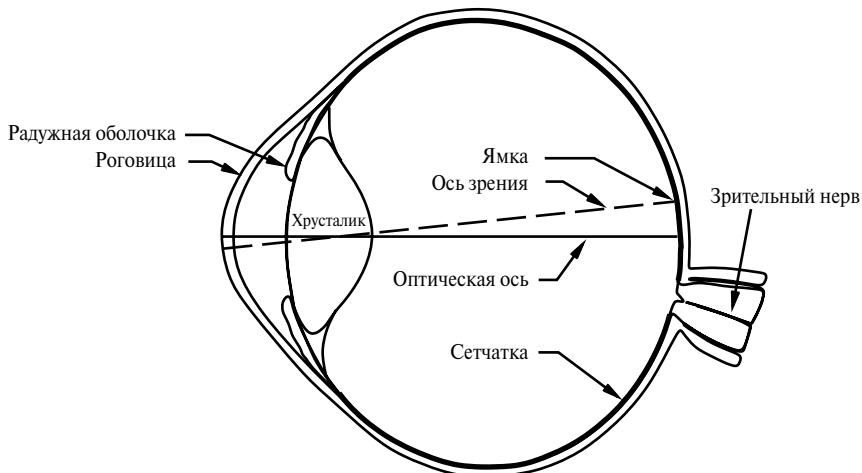


Рис. 24.2. Горизонтальный поперечный разрез человеческого глаза

### Свет: фотометрия формирования изображения

Свет — это наиболее важная предпосылка зрения; без света все изображения были бы одинаково темными, независимо от того, насколько интересной является сцена. **Фотометрия** — это наука о свете. Для наших целей мы создадим модель того, как свет в сцене преобразуется в интенсивность света на плоскости изображения, которая обозначается<sup>1</sup> как  $I(x, y)$ . Такая модель лежит в основе любой системы зрения, позволяющей выявлять по данным об интенсивности света на изображениях свойства внешнего мира. На рис. 24.3 показано оцифрованное изображение степлера на столе, а также отмечен квадратом блок пикселов с размерами  $12 \times 12$ , выделенный из изображения степлера. Работа любой компьютерной программы, предназначеннной для интерпретации изображения, начинается с матрицы значений эффективности, подобной этой.

Яркость пикселя на изображении пропорциональна количеству света, направленного в камеру от конечной части поверхности, ограниченной замкнутой кривой, в сцене, которая проектируется на данный пиксель. Это значение, в свою очередь, зависит от отражательных свойств рассматриваемой конечной части поверхности, а также от положения и распределения источников света в сцене. Кроме того, отражательные

<sup>1</sup> Если требуется также учитывать изменения интенсивности во времени, то используется выражение  $I(x, y, t)$ .

свойства зависят от остальной части сцены, поскольку другие поверхности в сцене могут стать косвенными источниками света, отражая падающий на них свет.

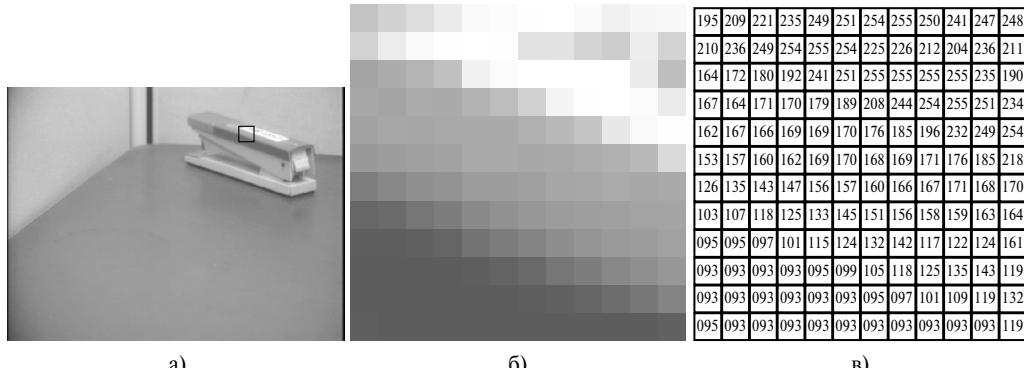


Рис. 24.3. Пример исходных данных для анализа изображения: фотография степлера на столе (а); блок пикселов размерами  $12 \times 12$ , взятый из изображения на рис. 24.3, а, в увеличенном виде (б); соответствующие значения яркости изображения, изменяющиеся в пределах от 0 до 255 (в)

Мы можем моделировать два различных вида отражения. **Зеркальное отражение** означает, что свет отражается от внешней поверхности объекта и подчиняется тому закону, что угол падения равен углу отражения. Так действует идеальное зеркало. **Диффузное отражение** означает, что свет проникает через поверхность объекта, поглощается объектом, а затем снова излучается за пределы его поверхности. Если поверхность является идеально рассеивающей (или **ламбертовой**), то свет рассеивается с равной интенсивностью во всех направлениях. Интенсивность зависит только от угла падения луча, поступающего от источника света: если источник света расположен прямо над поверхностью, то отражается больше всего света, а если лучи от источника света проходят почти параллельно поверхности, то количество отраженного света примерно равно нулю. Между этими двумя крайностями интенсивность отражения  $I$  подчиняется закону косинуса, установленному Ламбертом:

$$I = k I_0 \cos\theta$$

где  $I_0$  — интенсивность источника света;  $\theta$  — угол между источником света и перпендикуляром к поверхности;  $k$  — константа, называемая **отражательной способностью**, которая зависит от отражательных свойств поверхности. Она изменяется от 0 (для совершенно черных поверхностей) до 1 (для чисто белых поверхностей).

В действительности почти все поверхности обладают сочетанием свойств диффузного и зеркального отражения. Моделирование такого сочетания на компьютере — это квинтэссенция компьютерной графики. Прорисовка реалистических изображений обычно осуществляется по методу трассировки луча, цель которого заключается в моделировании физического процесса генерации света источниками света, а затем многократного повторного отражения.

### Цвет — спектрофотометрия формирования изображения

На рис. 24.3 мы представили черно-белое изображение, полностью игнорируя тот факт, что видимый свет складывается из целого ряда волн с разной длиной, начиная

от 400 нанометров (нм) на фиолетовом и заканчивая 700 нм на красном конце спектра. В некоторых случаях свет состоит из волн, имеющих лишь единственное значение длины, соответствующее одному из цветов радуги. Но в других случаях свет представляет собой комбинацию волн различной длины. Означает ли это, что в качестве меры для описанной выше величины  $I(x, y)$  необходимо использовать сочетание значений, а не единственное значение? Если бы нам требовалось точно представить физические свойства света, то действительно возникла бы указанная выше необходимость. Но если нам нужно лишь эмулировать процесс восприятия света людьми (и многими другими позвоночными), то можно пойти на компромисс. Эксперименты (начатые еще Томасом Юнгом в 1801 году) показали, что любая смесь световых волн с разными значениями длины, вне зависимости от ее сложности, может быть представлена в виде смеси, состоящей лишь из трех основных цветов. Это означает, что если есть генератор света, позволяющий составлять линейные комбинации световых волн с тремя значениями длины (как правило, для этого выбирают красный (700 нм), зеленый (546 нм) и синий (436 нм)), то путем регулировки рукояток для увеличения относительного содержания одного цвета и уменьшения другого можно составить любую комбинацию значений длин волн; по крайней мере, если полученная комбинация предназначена для восприятия человеком. Этот экспериментальный факт означает, что изображения могут быть представлены с помощью вектора, определяющего только три значения интенсивности в расчете на один пикセル, и каждое из этих значений должно соответствовать интенсивностям света с тремя основными значениями длины волны. На практике для воспроизведения изображения с высоким качеством достаточно отвести по одному байту для каждого значения. Правильность такого подхода к обеспечению трехцветного восприятия цвета подтверждается также тем, что в сетчатке имеются три типа колбочек, пиковое значение чувствительности которых находится в диапазоне значений длины волны соответственно 650, 530 и 430 нм. Однако возникающие при этом связи намного сложнее, чем можно было бы представить с помощью взаимно-однозначного отображения.

### 24.3. ОПЕРАЦИИ, ВЫПОЛНЯЕМЫЕ НА ПЕРВОМ ЭТАПЕ ОБРАБОТКИ ИЗОБРАЖЕНИЯ

Как было указано выше, свет, отражаясь от объектов в сцене, формирует изображение, состоящее, скажем, из пяти миллионов трехбайтовых пикселов. Как и при использовании датчиков всех других типов, полученный сигнал содержит шум, но в данном случае ситуация усугубляется тем, что объем полученных данных очень велик. В этом разделе будет показано, что можно сделать с данными изображения для того, чтобы упростить их обработку. Вначале рассмотрим операции сглаживания изображения, позволяющие уменьшить шум, а также операции, позволяющие обнаруживать края участков на изображении. Эти операции называются операциями “предварительной обработки” или операциями “низкого уровня”, поскольку они стоят на первом месте в конвейере операций. Визуальные операции предварительной обработки характеризуются тем, что выполняются локально (они могут применяться лишь к одному участку изображения без учета того, что есть еще какие-то другие участки изображения, пусть даже находящиеся на расстоянии всего нескольких пикселов), а также тем, что в них не требуются знания:

для сглаживания изображения и обнаружения краев не нужно задумываться над тем, какие объекты представлены на изображениях. Благодаря этому операции низкого уровня вполне могут быть реализованы с помощью параллельных обрабатывающих средств либо в живом организме, либо в электронном устройстве. Затем мы рассмотрим одну операцию среднего уровня — операцию сегментации изображения на участки. Операции на этом этапе все еще применяются к изображению, а не ко всей сцене, но уже появляются элементы нелокальной обработки.

Как было указано в разделе 15.2, под **сглаживанием** подразумевается предсказание значения переменной состояния в некоторый момент времени  $t$  в прошлом при наличии свидетельств, полученных, начиная с момента времени  $t$  и заканчивая всеми другими значениями времени вплоть до настоящего момента. Теперь мы применим такую же идею, но не во временной проблемной области, а в пространственной, и будем трактовать процесс сглаживания как предсказание значения данного конкретного пикселя, если известны значения окружающих его пикселов. Следует отметить, что необходимо четко понимать, каково различие между наблюдаемым значением, измеренным для какого-то пикселя, и истинным значением, которое в действительности должно было быть измерено в этом пикселе. Они могут быть разными из-за случайных ошибок измерений или из-за систематического отказа, поскольку может оказаться, что в этой точке неисправен элемент матрицы CCD.

Один из способов сглаживания изображения состоит в том, чтобы каждому пикселу присваивалось среднее значение характеристик его соседних пикселов. Такой способ обработки, как правило, исключает экстремальные значения. Но остается открытым вопрос: сколько нужно рассмотреть соседних пикселов — один, два или больше? Ответ на этот вопрос заключается в том, что для исключения гауссова шума следует рассчитать взвешенное среднее с использованием **фильтра с гауссовой характеристикой**. Напомним, что гауссова функция со среднеквадратичным отклонением  $\sigma$  выражается следующими формулами:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2} \quad \text{в одном измерении или}$$

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x^2+y^2)/2\sigma^2} \quad \text{в двух измерениях}$$

Под применением фильтра с гауссовой характеристикой подразумевается замена значения интенсивности  $I(x_0, y_0)$  суммой по всем  $(x, y)$  пикселям значений  $I(x, y) G_\sigma(d)$ , где  $d$  — расстояние от  $(x_0, y_0)$  до  $(x, y)$ . Такого рода взвешенная сумма применяется так часто, что для нее предусмотрено особое название и обозначение. Считается, что функция  $h$  представляет собой **свертку** двух функций,  $f$  и  $g$  (обозначается как  $h=f*g$ ), если имеют место следующие соотношения:

$$h(x) = \sum_{u=-\infty}^{+\infty} f(u) g(x-u) \quad \text{в одном измерении или}$$

$$h(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v) g(x-u, y-v) \quad \text{в двух измерениях}$$

Поэтому операция сглаживания осуществляется путем формирования свертки изображения и гауссова распределения,  $I * G_\sigma$ . Значение  $\sigma$ , равное 1 пикселу, является достаточным для сглаживания шума с небольшой интенсивностью. Если же значение  $\sigma$  соответствует 2 пикセルам, то происходит сглаживание шума с большей интенсивностью, но теряются некоторые мелкие детали. Поскольку влияние гауссова распределения уменьшается с расстоянием, на практике можно заменить пределы  $\pm\infty$  в суммах значениями, примерно равными  $\pm 3\sigma$ .

## Обнаружение краев

Следующая операция, выполняемая на первом этапе обработки изображения, состоит в обнаружении краев на плоскости изображения. **Краями** называются прямые или кривые линии на плоскости изображения, которые служат “водоразделом” для существенных изменений в яркости изображения. Целью обнаружения краев является повышение уровня абстракции и переход от перегруженного подробностями мультимегабайтового изображения к более компактному, абстрактному представлению, как показано на рис. 24.4. Необходимость в выполнении такой операции обусловлена тем, что линии краев на изображении соответствуют важным контурам в сцене. На этом рисунке приведены три примера изображений, на которых отмечены так называемые *сосредоточенные неоднородности*: сосредоточенная неоднородность по глубине, обозначенная меткой 1; сосредоточенные неоднородности по ориентации двух поверхностей, обозначенные меткой 2; сосредоточенная неоднородность по коэффициенту отражения, обозначенная меткой 3; сосредоточенная неоднородность по освещенности (тень), обозначенная меткой 4. Результаты операции обнаружения краев относятся только к данному конкретному изображению и поэтому не содержат данных, позволяющих распознать эти различные типы сосредоточенных неоднородностей в сцене, но такая задача может быть решена в ходе дальнейшей обработки.

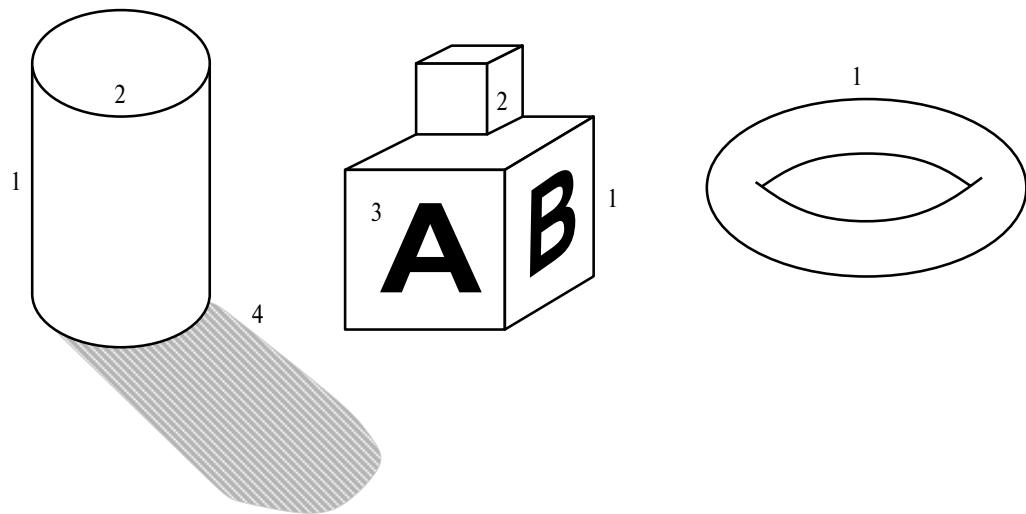


Рис. 24.4. Различные виды краев: сосредоточенные неоднородности по глубине (1); сосредоточенные неоднородности по ориентации поверхностей (2); сосредоточенные неоднородности по коэффициенту отражения (3); сосредоточенные неоднородности по освещенности (тени) (4)

На рис. 24.5, *a* приведено изображение сцены, на котором показан степлер, находящийся на столе, а на рис. 24.5, *б* приведены результаты применения алгоритма обнаружения краев к этому изображению. Вполне очевидно, что эти результаты весьма отличаются от идеального контурного рисунка. Небольшие края, соответствующие деталям изображения, не всегда выровнены друг с другом; на некоторых участках, где эти края обрываются, имеются пропуски, кроме того, есть и края, появившиеся под воздействием шума, которые не соответствуют каким-либо значимым деталям в этой сцене. Все указанные ошибки должны быть исправлены на последующих этапах обработки. Теперь перейдем к описанию того, как происходит обнаружение краев на изображении. Рассмотрим профиль яркости изображения вдоль одномерного поперечного разреза, перпендикулярного к некоторому краю (например, между левой стороной стола и стеной). Он выглядит примерно так, как показано на рис. 24.6, *а*. Местонахождение края соответствует значению  $x=50$ .

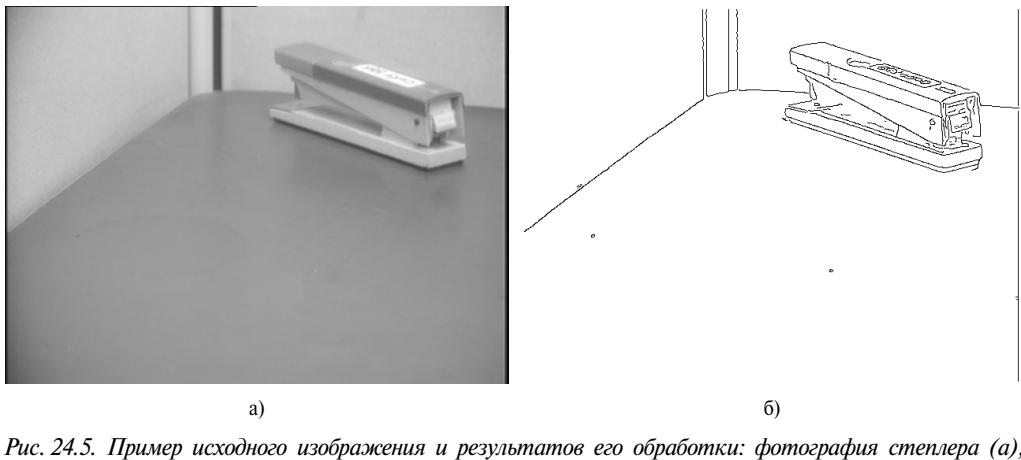


Рис. 24.5. Пример исходного изображения и результатов его обработки: фотография степлера (*а*); края, обнаруженные в результате обработки изображения на рис. 24.5, *а* (*б*)

Поскольку края соответствуют тем участкам изображения, где яркость подвергается резким изменениям, то на первый взгляд может показаться, что достаточно дифференцировать изображение и найти такие места, где производная  $I'(x)$  принимает большое значение. И действительно, такой подход в определенной степени осуществим. Тем не менее, как показано на рис. 24.6, *б*, кроме основного пикового значения с координатой  $x=50$ , обнаруживаются также дополнительные пиковые значения в других местах (например, с координатой  $x=75$ ), которые могут быть ошибочно приняты за настоящие края. Эти дополнительные пиковые значения возникают из-за наличия шума в изображении. Если вначале будет проведено сглаживание изображения, то количество фиктивных пиков уменьшится, как показано на рис. 24.6, *в*.

Теперь у нас появляется возможность применить на данном этапе определенную оптимизацию, поскольку операции сглаживания и поиска краев можно объединить в одну операцию. Существует теорема, согласно которой для любых функций  $f$  и  $g$  производная свертки,  $(f*g)'$ , равна свертке с производной,  $f*(g')'$ . Поэтому вместо сглаживания изображения с последующим дифференцированием можно просто выполнить свертку изображения с производной гауссовой функцией сглаживания,

$G_\sigma'$ . Таким образом, алгоритм поиска края в одном измерении может быть представлен следующим образом.

1. Выполнить свертку изображения  $I$  с функцией  $G_\sigma'$  для получения результатов свертки  $R$ .

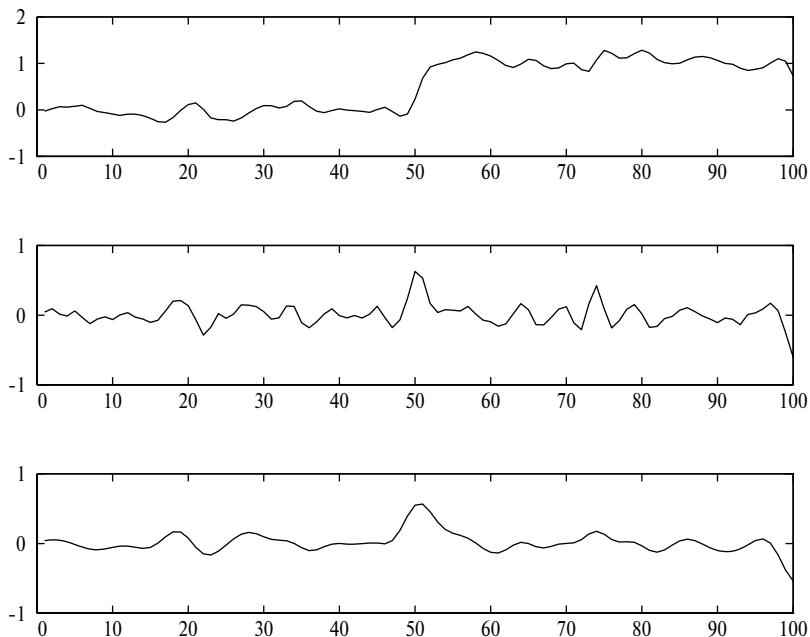


Рис. 24.6. Верхний рисунок: профиль интенсивности  $I(x)$  вдоль одномерного разреза, пересекающего край, который характеризуется ступенчатым изменением яркости. Средний рисунок: производная интенсивности,  $I'(x)$ . Большие значения на этом графике соответствуют краям, но представленные на нем данные содержат шум. Нижний рисунок: производная от сглаженной версии данных об интенсивности,  $(I * G_\sigma')'$ , которая может быть вычислена в одном шаге как свертка  $I * G_\sigma'$ . Исчез появившийся под воздействием шума пик с координатой  $x=75$ , который в других условиях рассматривался бы как признак наличия края

2. Обозначить как края те пиковые значения в  $| | R(x) | |$ , которые превышают некоторое заранее заданное пороговое значение  $T$ . Это пороговое значение выбирается таким образом, чтобы можно было устраниТЬ фиктивные пиковые значения, возникшие под воздействием шума.

В двух измерениях края могут проходить под любым углом  $\theta$ . Для обнаружения вертикальных краев можно применить такую очевидную стратегию: выполнить свертку с  $G_\sigma'(x) G_\sigma(y)$ . В направлении  $y$  эффект этой операции сводится к тому, что будет выполнено только сглаживание (под воздействием гауссовой свертки), а в направлении  $x$  результатом операции станет то, что дифференцирование будет сопровождаться сглаживанием. Поэтому алгоритм обнаружения вертикальных краев состоит в следующем.

1. Выполнить свертку изображения  $I(x, y)$  с функцией  $f_v(x, y) = G_\sigma'(x) G_\sigma(y)$  для получения результатов свертки  $R_v(x, y)$ .
2. Обозначить как края те пиковые значения в  $\|R_v(x, y)\|$ , которые превышают некоторое заранее заданное пороговое значение  $T$ .

Для того чтобы обнаружить какой-либо край, имеющий произвольную ориентацию, необходимо выполнить свертку изображения с двумя фильтрами,  $f_v = G_\sigma'(x) G_\sigma(y)$  и  $f_h = G_\sigma'(y) G_\sigma(x)$ , где функция  $f_h$  соответствует функции  $f_v$ , график которой повернут на  $90^\circ$ . Таким образом, алгоритм обнаружения краев, имеющих произвольную ориентацию, состоит в следующем.

1. Выполнить свертку изображения  $I(x, y)$  с функциями  $f_v(x, y)$  и  $f_h(x, y)$  для получения соответственно результатов свертки  $R_v(x, y)$  и  $R_h(x, y)$ . Определить  $R(x, y) = R_v^2(x, y) + R_h^2(x, y)$ .
2. Обозначить как края те пиковые значения в  $\|R(x, y)\|$ , которые превышают некоторое заранее заданное пороговое значение  $T$ .

После того как с помощью этого алгоритма будут отмечены пиксели краев, на следующем этапе необходимо связать друг с другом те пиксели, которые принадлежат к одним и тем же кривым краев. Такую операцию можно выполнить, приняв предположение, что любые два соседних пикселя, которые являются пикселями края с совместными ориентациями, должны принадлежать к одной и той же кривой края. Описанный выше процесс получил название процесса **обнаружения края Кэнни** в честь его разработчика Джона Кэнни (John Canny).

Края после их обнаружения становятся основой для многих этапов последующей обработки: их можно использовать для выполнения стереооптической обработки, обнаружения движения или распознавания объектов.

## Сегментация изображения

Мозг человека не использует полученные им результаты восприятия в непосредственном виде, а организует эти результаты определенным образом, поэтому вместо коллекции значений яркости, связанных с отдельными фоторецепторами, мозг выделяет целый ряд визуальных групп, которые обычно ассоциируются с объектами или частями объектов. Такая способность является не менее важной и для машинного зрения.

**Сегментация** — это процесс разбиения изображения на группы с учетом подобия характеристик пикселов. Основная идея этого процесса состоит в следующем: каждый пикセル изображения может быть связан с некоторыми визуальными свойствами, такими как яркость, цвет и текстура<sup>2</sup>. В пределах одного объекта или одной части объекта эти атрибуты изменяются относительно мало, тогда как при переходе через границу от одного объекта к другому обычно происходит существенное изменение одного или другого из этих атрибутов. Необходимо найти вариант разбиения изображения на такие множества пикселов, что указанные ограничения удовлетворяются в максимально возможной степени.

---

<sup>2</sup> Анализ свойств текстуры базируется на статистических данных, полученных применительно к небольшому замкнутому участку поверхности, центром которого является рассматриваемый пикSEL.

Существует целый ряд различных способов, с помощью которых эта интуитивная догадка может быть формализована в виде математической теории. Например, в [1402] рассматриваемая задача представлена как задача сегментации графа. Узлы графа соответствуют пикселям, а ребра — соединениям между пикселями. Ребрам, соединяющим пары пикселов  $i$  и  $j$ , присваиваются веса  $W_{ij}$  с учетом того, насколько близки значения яркости, цвета, текстуры и т.д. для двух пикселов соответствующей пары. Затем осуществляется поиск разбиений, которые минимизируют нормализованный критерий отсечения. Грубо говоря, критерием сегментации графа является критерий минимизации суммы весов соединений между группами и максимизации суммы весов соединений в пределах групп.

Процесс сегментации, основанный исключительно на использовании низкоуровневых локальных атрибутов, таких как яркость и цвет, чреват существенными ошибками. Чтобы надежно обнаруживать границы, связанные с объектами, необходимо также использовать высокоуровневые знания о том, какого рода объекты могут по всей вероятности встретиться в данной сцене. При распознавании речи такая возможность появилась благодаря использованию формальных средств скрытой марковской модели; в контексте обработки изображений поиск такой универсальной инфраструктуры остается темой интенсивных исследований. Так или иначе представление высокоуровневых знаний об объектах является темой следующего раздела.

## 24.4. ИЗВЛЕЧЕНИЕ ТРЕХМЕРНОЙ ИНФОРМАЦИИ

В данном разделе будет показано, как перейти от двухмерного изображения к трехмерному представлению сцены. Для нас важно перейти именно к стилю рассуждений о сцене в связи с тем, что агент в конечном итоге существует в мире, а не на плоскости изображения, а зрение предназначено для получения возможности взаимодействовать с объектами в том мире, где существует агент. Тем не менее для большинства агентов требуется только ограниченное абстрактное представление некоторых аспектов сцены, а не все подробности. Алгоритмы, используемые при решении задач взаимодействия с окружающим миром, которые были приведены в последних частях данной книги, распространяются на краткие описания объектов, а не на исчерпывающие перечисления каждой трехмерной конечной части поверхности, ограниченной замкнутой кривой.

Вначале в этом разделе рассматривается процесс **распознавания объекта**, в котором характеристики изображения (такие как края) преобразуются в модели известных объектов (таких как стеллеры). Распознавание объекта происходит в три этапа: сегментация сцены с выделением различных объектов, определение позиции и ориентации каждого объекта относительно наблюдателя и определение формы каждого объекта.

Определение позиции и ориентации объекта относительно наблюдателя (так называемой **позы** объекта) является наиболее важной операцией с точки зрения решения задач манипулирования и навигации. Например, чтобы робот мог передвигаться по полу заводского цеха в условиях ограниченного маневра, он должен знать местонахождение всех препятствий, чтобы иметь возможность спланировать путь, позволяющий избежать столкновения с ними. Если же робот должен выбрать и захватить какой-то объект, то он должен знать расположение этого объекта относи-

тельно манипулятора, чтобы выработать подходящую траекторию движения. Действия по манипулированию и навигации обычно осуществляются в рамках заданного контура управления, а сенсорная информация предоставляет обратную связь для модификации движения робота или перемещения манипулятора робота.

Представим позицию и ориентацию в математических терминах. Позиция точки  $P$  в сцене характеризуется тремя числами — координатами ( $X, Y, Z$ ) точки  $P$  в системе координат с началом координат в микроотверстии и с осью  $Z$ , проходящей вдоль оптической оси (см. рис. 24.1). В нашем распоряжении имеется перспективная проекция точки на изображении ( $x, y$ ). Эта проекция определяет луч, проходящий из микроотверстия, на котором расположена точка  $P$ ; неизвестным является расстояние. Термин “ориентация” может использоваться в двух описанных ниже смыслах.

1. Ориентация объекта как единого целого. Она может быть задана в терминах трехмерного вращения, связывающего систему координат этого объекта с системой координат камеры-обскуры.
2. Ориентация поверхности объекта в точке  $P$ . Она может быть задана с помощью нормального вектора  $\mathbf{n}$ , т.е. вектора, задающего направление, перпендикулярное к поверхности. Для представления ориентации поверхности часто используются переменные  $\angle$  угол поворота и  $\angle$  угол наклона. Углом поворота называется угол между осью  $Z$  и вектором  $\mathbf{n}$ , а углом наклона — угол между осью  $X$  и проекцией вектора  $\mathbf{n}$  на плоскость изображения.

По мере перемещения камеры-обскуры по отношению к объекту изменяются и расстояние до объекта, и его ориентация. Сохраняется только  $\angle$  форма объекта. Если объект представляет собой куб, он остается таковым и после его перемещения. В геометрии попытки формализовать понятие геометрической формы предпринимались в течение многих столетий; в конечном итоге было сформулировано такое основное понятие, что формой является то, что остается неизменным после применения некоторой группы преобразований, например сочетаний поворотов и переносов. Сложность заключается в том, что нужно найти способ представления глобальной формы, достаточно общий для того, чтобы с его помощью можно было описать широкий перечень объектов реального мира (а не только такие простые формы, как цилиндры, конусы и сферы) и при этом предусмотреть возможность легко восстанавливать информацию о форме из визуальных входных данных. Но гораздо лучше изучена проблема описания локальной формы поверхности. По сути, это может быть выполнено в терминах кривизны — определения того, как изменяется положение нормального вектора по мере передвижения в различных направлениях по этой поверхности. Если поверхность представляет собой плоскость, то положение нормального вектора вообще не изменяется. В случае цилиндрической поверхности при перемещении параллельно оси изменения не происходят, а при перемещении в перпендикулярном направлении вектор, нормальный к поверхности, вращается со скоростью, обратно пропорциональной радиусу цилиндра, и т.д. Все эти явления исследуются в научной области, называемой *дифференциальной геометрией*.

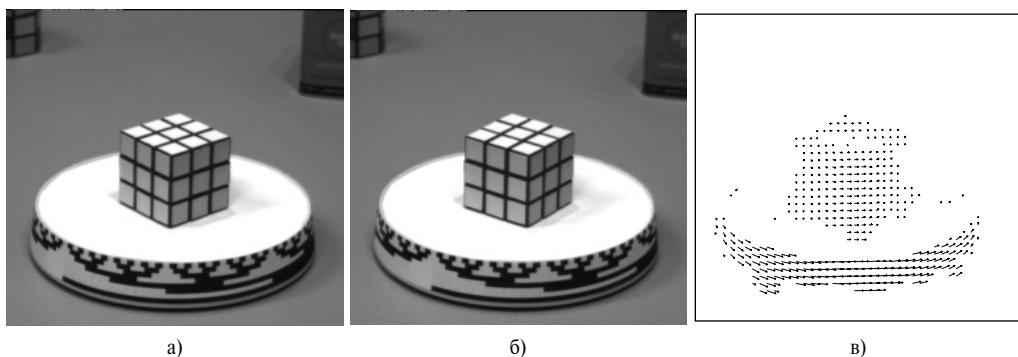
Форму объекта важно знать при выполнении некоторых задач манипулирования (например, чтобы определить, в каком месте лучше всего захватить данный объект), но наиболее значительную роль форма объекта играет при распознавании объектов. При решении последней задачи наиболее значащими подсказками, позволяющими

идентифицировать объекты, определять с помощью методов классификации, что это изображение является примером какого-то класса, встречавшегося ранее, и т.д., служат геометрическая форма наряду с цветом и текстурой.

Фундаментальный вопрос состоит в следующем: “Если дано, что во время создания перспективной проекции все точки в трехмерном мире, расположенные вдоль одного луча, проходящего через микроотверстие, спроектировались на одну и ту же точку в изображении, то как теперь восстановить трехмерную информацию?” В визуальных стимулах, относящихся к числу применимых для этой цели, содержится целый ряд характерных признаков, включая **движение, бинокулярные стереоданные, текстуру, затенение и контуры**. Каждый из этих характерных признаков позволяет опереться на исходные предположения о физических сценах, чтобы можно было получить (почти) полностью непротиворечивые интерпретации этих сцен. Каждый из указанных характерных признаков рассматривается в пяти приведенных ниже разделах.

### Движение

До сих пор рассматривалось только одно изображение одновременно. Но видеокамеры позволяют получать 30 кадров в секунду и различия между кадрами могут стать важным источником информации. Если видеокамера движется относительно трехмерной сцены, то в изображении возникает кажущееся результирующее движение, называемое **оптическим потоком**. Оптический поток содержит информацию о направлении и скорости движения характеристик в изображении, являющегося результатом относительного движения между наблюдателем и сценой. На рис. 24.7, а, б показаны два кадра из видеофильма о вращении кубика Рубика. На рис. 24.7, в показаны векторы оптического потока, вычисленные на основании этих двух изображений. В оптическом потоке зашифрована полезная информация о структуре сцены. Например, при наблюдении из движущегося автомобиля удаленные объекты характеризуются гораздо более медленным кажущимся движением по сравнению с близкими объектами, поэтому скорость кажущегося движения позволяет получить определенную информацию о расстоянии.



*Рис. 24.7. Пример использования понятия оптического потока: кубик Рубика на поворотном столе, приведенном во вращение (а); тот же кубик, показанный через 19/30 секунды (любезно предоставлено Ричардом Шелински (Richard Szeliski)) (б); векторы потока, вычисленные путем сравнения двух изображений, приведенных на рис. 24.7, а, б (любезно предоставлено Джо Вебером (Joe Weber) и Джитендрай Маликом (Jitendra Malik)) (в)*

Поле вектора оптического потока может быть представлено с помощью его компонентов  $v_x(x, y)$  в направлении  $x$  и  $v_y(x, y)$  в направлении  $y$ . Для измерения оптического потока необходимо найти соответствующие точки между одним временным кадром и следующим. При этом используется тот факт, что замкнутые участки изображения, сосредоточенные вокруг соответствующих точек, характеризуются аналогичными шаблонами интенсивности. Рассмотрим блок пикселов с центром в пикселе  $p$ , в точке  $(x_0, y_0)$ , во время  $t_0$ . Этот блок пикселов необходимо сравнить с блоками пикселов, центрами которых являются различные потенциально применимые пиксели  $q_i$  с координатами  $(x_0+D_x, y_0+D_y)$  во время  $t_0+D_t$ . Одним из возможных критериев подобия является **сумма квадратов разностей** (Sum of Squared Differences — SSD):

$$SSD(D_x, D_y) = \sum_{(x, y)} (\mathcal{I}(x, y, t) - \mathcal{I}(x+D_x, y+D_y, t+D_t))^2$$

Здесь координаты  $(x, y)$  принимают свои значения среди пикселов в блоке с центром в точке  $(x_0, y_0)$ . Найдем значения  $(D_x, D_y)$ , которые минимизируют выражение для SSD. В таком случае оптический поток в точке  $(x_0, y_0)$  принимает значение  $(v_x, v_y) = (D_x/D_t, D_y/D_t)$ . Еще один вариант состоит в том, что можно максимизировать **взаимную корреляцию** следующим образом:

$$Correlation(D_x, D_y) = \sum_{(x, y)} \mathcal{I}(x, y, t) \mathcal{I}(x+D_x, y+D_y, t+D_t)$$

Метод с использованием взаимной корреляции действует лучше всего, если сцена характеризуется наличием текстуры, в результате чего блоки пикселов (называемые также *окнами*) содержат значительные вариации яркости среди входящих в них пикселов. Если же рассматривается ровная белая стена, то взаимная корреляция обычно остается почти одинаковой для различных потенциальных согласований  $q$  и алгоритм сводится к операции выдвижения слепого предположения.

Допустим, что наблюдатель движется с линейной скоростью (или скоростью переноса)  $T$  и с угловой скоростью  $\omega$  (таким образом, эти параметры описывают **самодвижение**). Можно вывести уравнение, связывающее скорости наблюдателя, оптический поток и положения объектов в сцене. Если предположить, что  $f=1$ , то из этого следуют уравнения

$$\begin{aligned} v_x(x, y) &= \left[ -\frac{T_x}{Z(x, y)} - \omega_y + \omega_z Y \right] - x \left[ -\frac{T_z}{Z(x, y)} - \omega_x Y + \omega_y X \right] \\ v_y(x, y) &= \left[ -\frac{T_y}{Z(x, y)} - \omega_z X + \omega_x \right] - y \left[ -\frac{T_z}{Z(x, y)} - \omega_x Y + \omega_y X \right] \end{aligned}$$

где  $Z(x, y)$  задает координату  $z$  точки в сцене, соответствующей точке на изображении с координатами  $(x, y)$ .

Достаточно хорошего понимания того, что при этом происходит, можно достичь, рассмотрев случай чистого переноса. В таком случае выражения для поля потока принимают следующий вид:

$$v_x(x, y) = \frac{-T_x + x T_z}{Z(x, y)}, \quad v_y(x, y) = \frac{-T_y + y T_z}{Z(x, y)}$$

Теперь становятся очевидными некоторые интересные свойства. Оба компонента оптического потока,  $v_x(x, y)$  и  $v_y(x, y)$ , принимают нулевое значение в точке с координатами  $x=T_x/T_z$ ,  $y=T_y/T_z$ . Эта точка называется **фокусом расширения** поля потока. Предположим, что мы изменим начало координат в плоскости  $x-y$  для того, чтобы оно находилось в фокусе расширения; в таком случае выражение для оптического потока принимает особенно простую форму. Допустим, что  $(x', y')$  — это новые координаты, определяемые соотношениями  $x'=x-T_x/T_z$ ,  $y'=y-T_y/T_z$ . В таком случае становятся справедливыми следующие уравнения:

$$v_x(x', y') = \frac{x' T_z}{Z(x', y')}, \quad v_y(x', y') = \frac{y' T_z}{Z(x', y')}$$

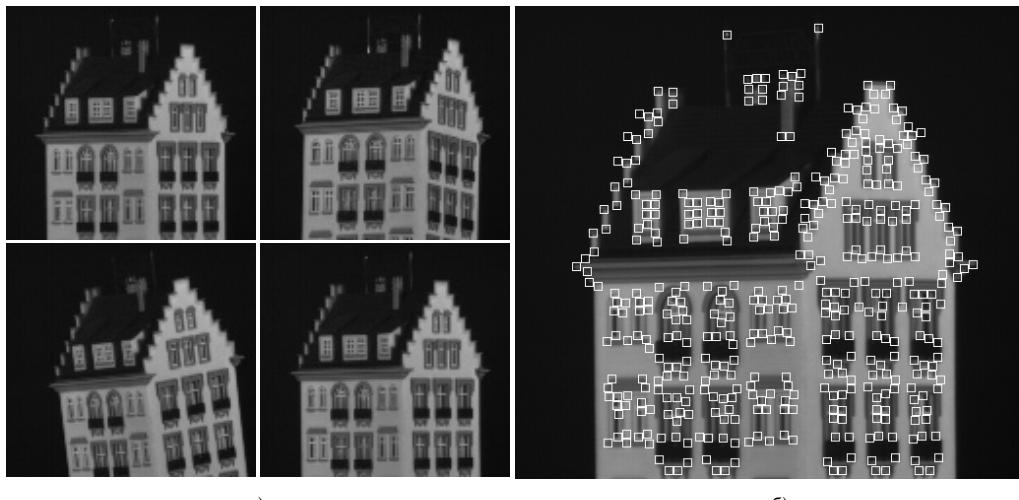
Эти уравнения имеют некоторые интересные области применения. Предположим, что летящая муха пытается сесть на стену и хочет определить, в какой момент она коснется стены, если будет сохраняться текущая скорость. Это время задается термом  $Z/T_z$ . Следует отметить, что мгновенное значение поля оптического потока не позволяет получить ни данные о расстоянии  $Z$ , ни данные о компоненте скорости  $T_z$ , но вместе с тем предоставляет значение соотношения этих двух параметров и поэтому может использоваться для управления приближением к поверхности посадки. Эксперименты, проведенные над мухами в реальных условиях, показали, что указанные насекомые действуют именно по этому принципу. Мухи относятся к числу наиболее ловких летающих объектов среди всех живых существ и машин, а интереснее всего то, что они добиваются этого с помощью системы зрения, имеющей невероятно низкое пространственное разрешение (поскольку в глазу мухи имеется всего около 600 рецепторов, тогда как в глазу человека — порядка 100 миллионов), но блестящее временное разрешение.

Чтобы получить данные о глубине, необходимо воспользоваться несколькими кадрами. Если видеокамера направлена на твердое тело, то его форма не изменяется от кадра к кадру и поэтому появляется возможность лучше решать задачу измерения оптического потока, неизменно подверженного шуму. Результаты применения одного из подобных подходов, полученные Томази и Канаде [1511], показаны на рис. 24.8 и 24.9.

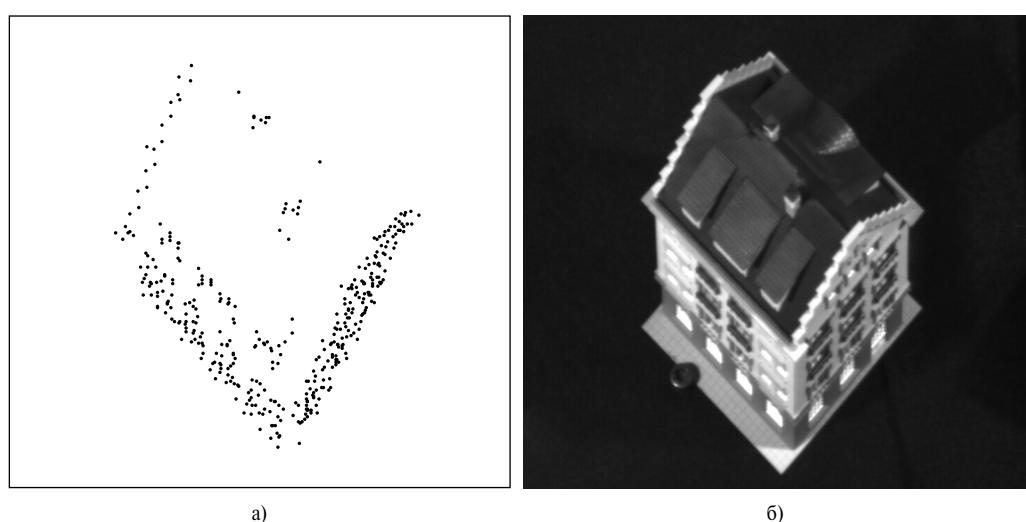
### Бинокулярные стереоданные

Большинство позвоночных имеют два глаза. Это не только удобно в качестве резерва на случай потери одного глаза, но и дает также некоторые другие преимущества. У многих травоядных глаза расположены по обе стороны головы, что позволяет им иметь более широкий обзор. С другой стороны, у хищников глаза находятся в передней части головы, благодаря чему они используют **бинокулярные стереоданные**. Принцип бинокулярного зрения аналогичен принципу, в котором используется параллакс, возникающий во время движения, за тем исключением, что вместо использования изображений, сменяющихся во времени, применяются два изображения (или, в случае систем машинного зрения, и большее количество изображений), которые разделены в пространстве; например, такие изображения передают в мозг глаза человека, смотрящего прямо вперед. Поскольку данная конкретная характеристика в сцене будет находиться в различных местах по отношению к оси  $z$  каждой плоскости изображения, то после совмещения двух изображений будет обнаруживаться

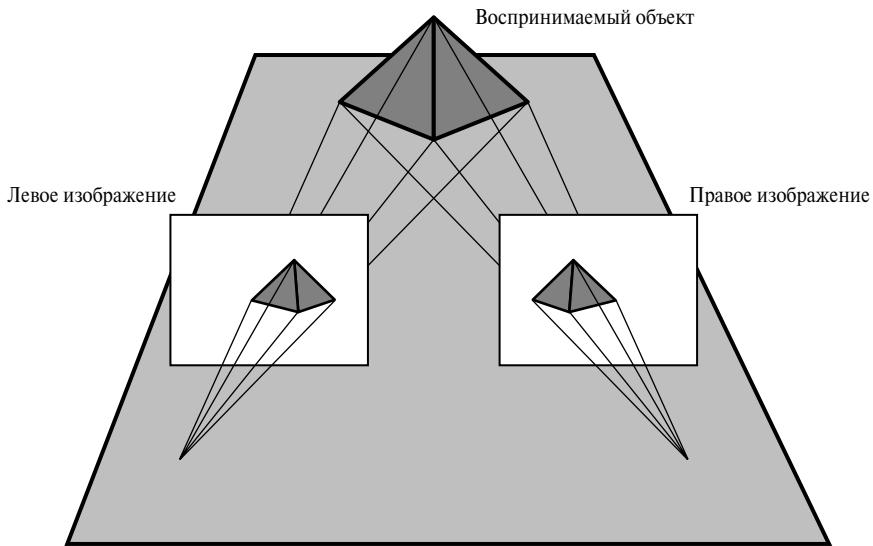
❖ **рассогласование** в местонахождениях этой характеристики в двух изображениях. Такое явление можно наблюдать на рис. 24.10, где ближайшая точка пирамиды сдвинута влево на правом изображении и вправо на левом изображении.



*Рис. 24.8. Пример применения нескольких кадров: четыре кадра из видеопоследовательности, в которой камера передвигается и вращается относительно объекта (а); первый кадр из последовательности, обозначенный небольшими прямоугольниками, подчеркивающими характерные особенности, которые были обнаружены детектором характеристик (любезно представлено Карлом Томази (Carlo Tomasi)) (б)*



*Рис. 24.9. Пример результатов обработки нескольких кадров: трехмерная реконструкция местонахождений характеристик изображения, показанных на рис. 24.8 (вид сверху) (а); настоящий дом, снимок которого получен из той же позиции (б)*



*Рис. 24.10. Принцип формирования стереоданных: изменение положения видеокамеры приводит к получению немного отличающихся друг от друга двухмерных представлений одной и той же трехмерной сцены*

Проанализируем геометрические соотношения между рассогласованием и глубиной. Прежде всего рассмотрим случай, в котором оба глаза (или обе видеокамеры) направлены прямо вперед, поэтому их оптические оси параллельны. В таком случае связь между изображением в правой и в левой видеокамере состоит в том, что для преобразования одного изображения в другое достаточно выполнить сдвиг вдоль оси  $x$  на величину  $b$  (расстояние между камерами, или опорная линия). Для вычисления рассогласований по горизонтали и вертикали, которые выражаются в виде  $H=v_x\Delta t$ ,  $V=v_y\Delta t$ , можно использовать уравнения оптического потока из предыдущего раздела, при условии, что  $T_x=b/\Delta t$  и  $T_y=T_z=0$ . Параметры вращательного сдвига  $\omega_x$ ,  $\omega_y$  и  $\omega_z$  равны нулю. Таким образом,  $H=b/Z$ ,  $V=0$ . Иными словами, рассогласование по горизонтали равно отношению длины опорной линии к глубине, а рассогласование по вертикали равно нулю.

При нормальных условиях наблюдения люди ~~фиксируют~~ фиксируют свой взгляд; это означает, что они выбирают в качестве объекта наблюдения некоторую точку в сцене, и в этой точке пересекаются оптические оси двух глаз. На рис. 24.11 показана ситуация, в которой взгляд человека, смотрящего двумя глазами, зафиксирован в точке  $P_0$ , находящейся на расстоянии  $Z$  от средней точки между глазами. Для удобства мы будем вычислять угловое рассогласование, измеряемое в радианах. Рассогласование в точке фиксации  $P_0$  равно нулю. Для некоторой другой точки  $P$  в сцене, которая находится дальше на величину  $\delta Z$ , можно вычислить угловые смещения левого и правого изображений точки  $P$ , которые будут обозначаться соответственно  $P_L$  и  $P_R$ . Если каждое из этих изображений смещено на угол  $\delta\theta/2$  относительно  $P_0$ , то смещение между  $P_L$  и  $P_R$ , которое представляет собой рассогласование в точке  $P$ , выражается величиной  $\delta\theta$ . Простые геометрические преобразования позволяют получить следующее выражение:

$$\frac{\delta\theta}{\delta Z} = \frac{-b}{Z^2}$$

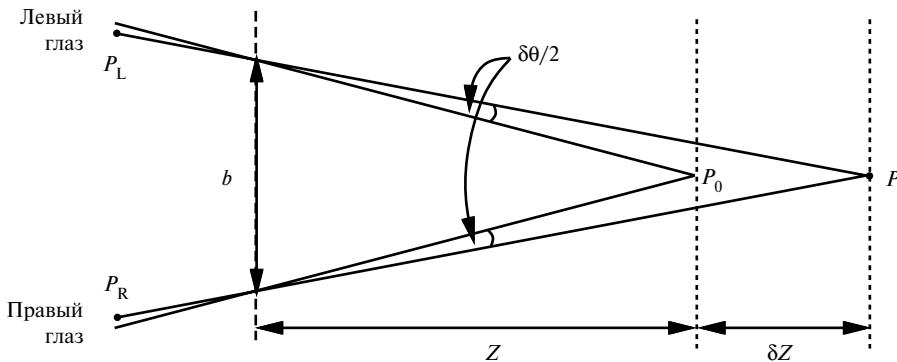


Рис. 24.11. Зависимость между рассогласованием и глубиной в стереоданных

У людей величина  $b$  (базисная линия) составляет около 6 см. Предположим, что расстояние  $Z$  примерно равно 100 см. В таком случае наименьшее доступное обнаружению значение  $\delta\theta$  (соответствующее размеру светочувствительного элемента глаза) составляет около 5 угловых секунд, а это означает, что разность расстояний  $\delta Z$  равна 0,4 мм. При  $Z=30$  см получим значение  $\delta Z=0,036$  мм, которое на удивление мало. Это означает, что на расстоянии 30 см люди способны различать значения глубины, которые отличаются друг от друга на столь малую величину, как 0,036 мм, что позволяет им продевать нитку через ушко иголки и выполнять другие тонкие операции.

### Градиенты текстуры

В повседневной речи под текстурой подразумевается свойство поверхностей, связанное с осязательными ощущениями, которое оно напоминает (слово “текстура” имеет тот же корень, что и слово “текстиль”). А в проблематике машинного зрения этим словом обозначается тесно связанное понятие, определяющее наличие повторяющегося в пространстве рисунка на поверхности, который может быть обнаружен визуально. В качестве примеров можно назвать одинаковый ряд окон на здании, стежки на свитере, пятна на шкуре леопарда, травинки на лужайке, гальку на берегу и толпу людей на стадионе. Иногда взаимное расположение повторяющихся рисунков является весьма регулярным, как в случае стежков на свитере, а в других случаях, таких как галька на берегу, регулярность обеспечивается только в определенном статистическом смысле: плотность расположения гальки приблизительно одинакова в разных частях пляжа.

Сказанное выше является справедливым применительно только к реальной сцене, а на изображении кажущиеся размеры, форма, взаиморасположение и другие характеристики элементов текстуры (называемых текселами) действительно различаются, как показано на рис. 24.12. Например, черепицы выглядят одинаковыми только в реальной сцене, а на изображении размеры и форма проекций черепиц варьируют, и это связано с двумя описанными ниже основными причинами.

1. Различия в расстояниях от текселов до видеокамеры. Напомним, что в перспективной проекции удаленные объекты кажутся меньшими. Коэффициент масштабирования равен  $1/Z$ .

2. Различия в ракурсах текстелов. Эта причина связана с ориентацией каждого текстела относительно линии зрения, направленной от камеры. Если текстел расположен перпендикулярно линии зрения, то ракурс отсутствует. Величина эффекта ракурса пропорциональна  $\cos\sigma$ , где  $\sigma$  — угол поворота плоскости текстела.

Проведя определенные выкладки в рамках математического анализа, можно вычислить выражения для скорости изменения различных характеристик текстел изображения, таких как площадь, ракурс и плотность. Такие показатели называются **градиентами текстуры** и являются функциями от формы поверхности, а также от углов ее поворота и наклона по отношению к местонахождению наблюдателя.

Для восстановления данных о форме из данных о текстуре можно использовать следующий двухэтапный процесс: а) измерить градиенты текстуры; б) определить оценочные значения формы поверхности, а также углов ее поворота и наклона, которые могли бы привести к получению измеренных градиентов текстуры. Результаты применения этого процесса приведены на рис. 24.12.



*Рис. 24.12. Примеры применения процесса восстановления формы: сцена, на которой показан градиент текстуры. Восстановление данных об ориентации поверхности может быть выполнено на основании предположения, что реальная текстура является однородной. Вычисленное значение ориентации поверхности показано путем наложения на изображение белого кружка и указателя, трансформированного так, как если бы кружок был нарисован на поверхности в этом месте (а); восстановление данных о форме по данным о текстуре в случае криволинейной поверхности (изображения любезно предоставлены Джитендрай Маликом (Jitendra Malik) и Рут Розенхольц (Ruth Rosenholtz) [972]) (б)*

## Затенение

Затенение (под этим подразумевается изменение интенсивности света, полученного от различных участков поверхности в сцене) определяется геометрией сцены и отражательными свойствами поверхностей. В компьютерной графике создание затенения сводится к вычислению значений яркости изображения  $I(x, y)$  с учетом геометрии сцены и отражательных свойств объектов в сцене. В проблематике машинного зрения решается обратная задача — восстановление данных о геометрии и отражательных свойствах по данным об яркости изображения  $I(x, y)$ . Как оказалось, эта задача с трудом поддается решению, за исключением самых простейших случаев.

Начнем с ситуации, в которой действительно можно найти решение задачи определения данных о форме на основании данных о затенении. Рассмотрим ламбертову поверхность, свет на которую падает от удаленного точечного источника света. Предположим, что поверхность находится достаточно далеко от видеокамеры, чтобы можно было использовать ортогональную проекцию в качестве аппроксимации перспективной проекции. Яркость изображения определяется с помощью следующей формулы:

$$I(x, y) = k \mathbf{n}(x, y) \cdot \mathbf{s}$$

где  $k$  — константа масштабирования;  $\mathbf{n}$  — единичный вектор, нормальный к поверхности;  $\mathbf{s}$  — единичный вектор, направленный в сторону источника света. Поскольку  $\mathbf{n}$  и  $\mathbf{s}$  — единичные векторы, их точечное произведение представляет собой косинус угла между ними. Форму поверхности можно определить, следя за тем, как изменяется направление нормального вектора  $\mathbf{n}$ , движущегося вдоль поверхности. Предположим, что значения  $k$  и  $\mathbf{s}$  известны. Поэтому задача сводится к тому, чтобы восстановить данные о векторе  $\mathbf{n}(x, y)$ , нормальном к поверхности, если известна интенсивность изображения  $I(x, y)$ .

Прежде всего необходимо отметить, что задача определения  $\mathbf{n}$ , если дана яркость  $I$  в данном пикселе  $(x, y)$ , не разрешима локально. Может быть вычислен угол, под которым вектор  $\mathbf{n}$  пересекается с вектором, направленным к источнику света, но полученный результат позволяет лишь узнать, что этот вектор находится в определенном конусе направлений с осью  $\mathbf{s}$  и углом от вершины  $\theta = \cos^{-1}(I/k)$ . Чтобы перейти к более точным вычислениям, необходимо отметить, что вектор  $\mathbf{n}$  не может изменяться произвольно при переходе от пикселя к пикселю. Он соответствует нормальному вектору гладкой конечной части поверхности, ограниченной замкнутой кривой, и поэтому также должен изменяться плавно (формальным названием для этого ограничения является [интегрируемость](#)). На основе этой идеи было разработано несколько различных методов. Один из них состоит в том, что нужно использовать другое выражение для  $\mathbf{n}$ , в терминах частных производных  $Z_x$  и  $Z_y$  глубины  $Z(x, y)$ . Такой подход приводит к получению частного дифференциального уравнения для  $Z$ , которое может быть решено для получения данных о глубине  $Z(x, y)$  с учетом подходящих граничных условий.

Этот подход может быть немного обобщен. Например, не обязательно, чтобы поверхность была ламбертовой, а источник света был точечным. При условии, что существует возможность вычислить [карту коэффициентов отражения](#)  $R(\mathbf{n})$ , которая задает значения яркости конечной части поверхности, ограниченной замкнутой кривой, как функции от положения нормального вектора  $\mathbf{n}$  к этой поверхности, могут по сути применяться методы такого же типа.

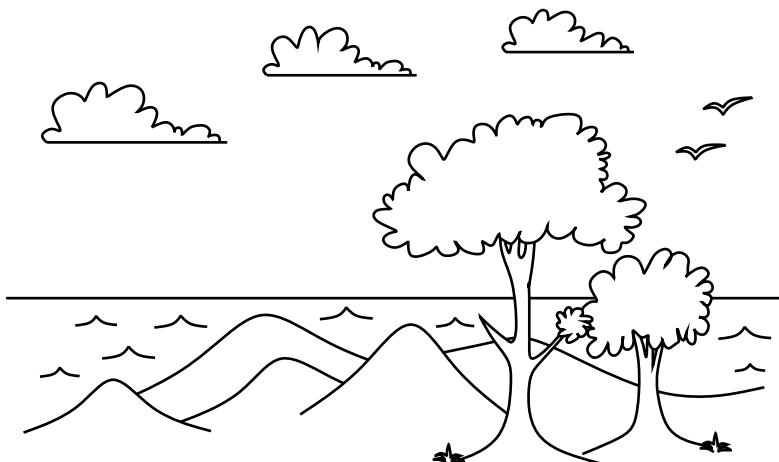
Настоящие сложности возникают, когда приходится иметь дело со взаимными отражениями. Если рассматривается типичная сцена внутри помещения, такая как сцена, в которой показаны объекты внутри офиса, то можно заметить, что поверхности освещаются не только источниками света, но и светом, отраженным от других поверхностей в сцене, которые фактически служат в качестве вторичных источников света. Такие эффекты взаимного освещения являются весьма значительными. В подобной ситуации формальный подход, предусматривающий использование карты коэффициентов отражения, становится полностью неприменимым, поскольку яр-

кость изображения зависит не только от положения нормального вектора к поверхности, но также и от сложных пространственных связей между различными поверхностями в сцене.

Не подлежит сомнению, что люди обладают определенными способностями к восприятию данных о форме на основании данных о затенении, поэтому указанная задача продолжает привлекать значительный интерес, несмотря на все сложности, связанные с ее решением.

## Контуры

Рассматривая контурный рисунок, подобный приведенному на рис. 24.13, мы получаем живое восприятие трехмерной формы и расположения. Благодаря чему это происходит? Ведь уже было сказано выше, что к получению одного и того же контурного рисунка приводят обработка не одной конфигурации сцены, а бесконечного количества таких конфигураций. Кроме того, следует отметить, что контурный рисунок позволяет даже получить представление о наклоне и повороте поверхностей. Такое ощущение может достигаться благодаря использованию сочетания знаний высокого уровня (знаний о типичных формах) с ограничениями низкого уровня.



*Рис. 24.13. Контурный рисунок, позволяющий получить полное представление о том, что на нем изображено (любезно предоставлен Айшой Маликом (Isha Malik))*

Рассмотрим, какие качественные знания могут быть получены с помощью контурного рисунка. Как было описано выше, линии на рисунке могут иметь много разных трактовок (см. рис. 24.4 и его подрисуночную подпись). Задача оценки фактической значимости каждой линии на изображении называется **разметкой линий**; она была одной из первых задач, изучаемых в области машинного зрения. На данный момент займемся изучением упрощенной модели мира, в которой объекты не имеют отметок на поверхности, а линии, обусловленные наличием сосредоточенных неоднородностей освещенности, такие как края теней и блики, были удалены на каком-то из этапов предварительной обработки, что позволяет нам сконцентрировать

свое внимание на контурных рисунках, где каждая линия соответствует сосредоточенной неоднородности либо по глубине, либо по ориентации.

В таком случае каждую линию можно отнести к одному из двух классов: рассматривать ее как проекцию **лимба** (геометрического места тех точек на поверхности, где луч зрения проходит по касательной к поверхности) или как **край** (поверхностная нормальная сосредоточенная неоднородность). Кроме того, каждый край может быть классифицирован как выпуклый, вогнутый или закрывающий (под этим подразумевается, что он закрывает то, что находится за ним). Что касается закрывающих краев и лимбов, то желательно иметь возможность определять, какая из двух поверхностей, примыкающих к кривой на контурном рисунке, является ближайшей к наблюдателю в данной сцене. Такие наложения линий могут быть представлены путем присваивания каждой линии одной из шести перечисленных ниже возможных **меток линий**, как показано на рис. 24.14.

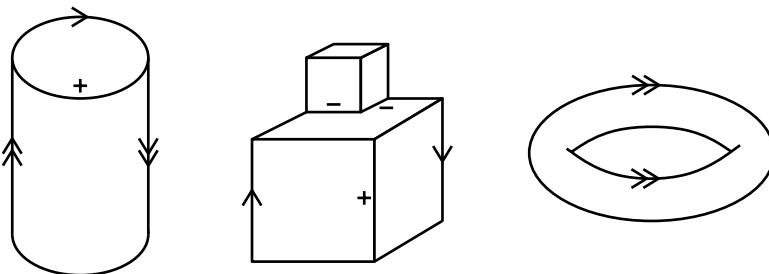


Рис. 24.14. Различные виды меток линий

1. Метки + и - представляют соответственно выпуклые и вогнутые края. Они связаны с поверхностными нормальными сосредоточенными неоднородностями, в которых видны обе поверхности, стыкующиеся вдоль этого края.
2. Метка ← или → представляет закрывающий выпуклый край. При просмотре сцены из видеокамеры обе конечные части поверхности, ограниченные замкнутой кривой, которые стыкуются вдоль этого края, лежат на одной и той же стороне, но одна из них закрывает другую. По мере перемещения по направлению стрелки эти закрывающие поверхности остаются справа.
3. Метка ←→ или →→ представляет лимб. На этой линии поверхность плавно искривляется по кругу, закрывая саму себя. По мере перемещения в направлении, обозначенным двойной стрелкой, закрывающая поверхность остается справа. Луч зрения проходит по касательной к поверхности во всех точках лимба. По мере изменения точки зрения лимбы меняют свое положение на поверхности объекта.

Если количество линий на рисунке равно  $n$ , то количество вариантов присваивания меток линий, определяемое законами комбинаторики, равно  $6^n$ , но количество физически возможных вариантов присваивания по сравнению с этим количеством составляет лишь очень небольшую величину. Задача определения таких возможных присваиваний меток называется *задачей разметки линий*. Следует отметить, что эта задача имеет смысл, только если метка остается одинаковой на всем протяжении линии. Но такое условие не всегда соблюдается, поскольку метки могут изменяться

вдоль линий на изображениях выпукло-вогнутых криволинейных объектов. В настоящей главе для предотвращения указанных сложностей будут рассматриваться исключительно только многогранные объекты.

Хаффмен [702] и Клоувс [271] независимо друг от друга впервые предприняли попытку применить систематический подход к анализу сцен с многогранными объектами. В своем анализе Хаффмен и Клоувс ограничивались сценами с непрозрачными ~~и~~ трехгранными твердыми телами; таковыми являются объекты, в которых в каждой вершине сходятся три и только три плоские поверхности. В случае наличия сцен с многочисленными объектами они, кроме этого, исключали такие выравнивания объектов, которые могли бы привести к нарушению предположения о наличии только трехгранных объектов, например сцен, в которых два куба имеют общий край. Не допускалось также наличие ~~и~~ трещин (т.е. “краев”, вдоль которых касательные плоскости являются непрерывными). Для такого мира трехгранных объектов Хаффмен и Клоувс подготовили исчерпывающий список всех различных типов вершин и описали всевозможные способы, с помощью которых эти вершины могут рассматриваться под общей точкой зрения. Условие, согласно которому должна существовать общая точка зрения, фактически гарантирует то, что если возникает небольшое движение глаза наблюдателя, ни одно из соединений плоскостей не меняет свой характер. Например, из этого условия следует, что если три линии пересекаются на изображении, то должны также пересекаться соответствующие края в сцене.

Четыре способа, с помощью которых три плоские поверхности могут быть соединены в одной вершине, показаны на рис. 24.15. Эти примеры могут быть также составлены путем разделения куба на восемь ~~и~~ октантов. В таком случае различные возможные трехгранные вершины в центре куба создаются путем заполнения разных октантов. Вершина, обозначенная цифрой 1, соответствует одному заполненному октанту, вершина с цифрой 3 — трем заполненным октантам и т.д. Рекомендуем читателям самим убедиться в том, что на данном рисунке действительно представлены все возможности. Например, попытка заполнить два октанта в кубе не приводит к созданию допустимой трехгранной вершины в центре. Следует также отметить, что эти четыре случая соответствуют различным комбинациям выпуклых и вогнутых краев, которые встречаются в данной вершине.

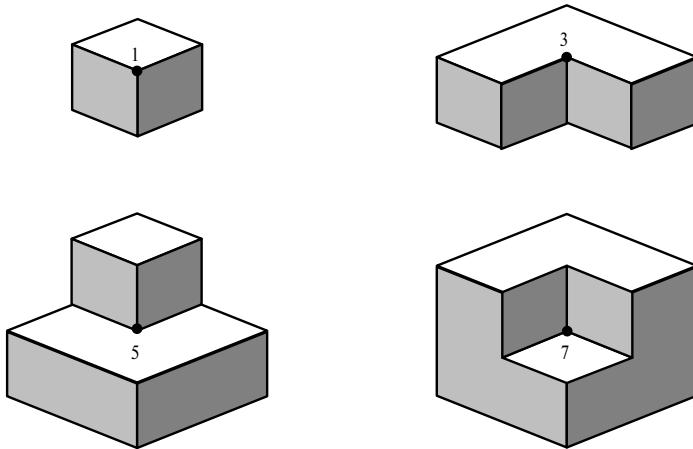


Рис. 24.15. Четыре вида трехгранных вершин

Три края, встречающихся в вершине, делят окружающее пространство на восемь октантов. Вершина видна из любого октанта, не заполненного твердым материалом. Перемещение точки зрения в пределах одного октанта не приводит к получению изображения с различными типами соединений. Вершина, обозначенная цифрой 1 на рис. 24.15, может рассматриваться из любого из оставшихся семи октантов; при этом наблюдаются метки соединения, показанные на рис. 24.16.

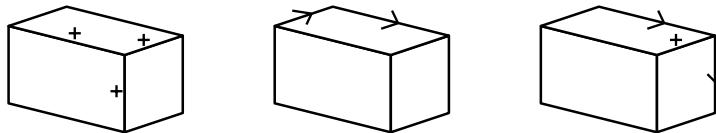


Рис. 24.16. Изменение внешнего вида вершины, обозначенной цифрой 1 на рис. 24.15

Работа по составлению исчерпывающего списка различных способов, с помощью которых может рассматриваться каждая вершина, привела к получению вариантов, показанных на рис. 24.17. Получено четыре различных типа соединений, которые могут быть выделены на изображении: L-соединения, Y-соединения, стреловидные соединения и Т-соединения. L-соединения соответствуют двум видимым краям. Y-соединения и стреловидные соединения соответствуют результатам рассмотрения трех краев, но различие между ними состоит в том, что в Y-соединении ни один из трех углов не превышает  $180^\circ$ . Т-соединения связаны с закрытием одной поверхности другой. Если ближайшая, непрозрачная поверхность закрывает вид на дальние расположенные ее край, будет получен непрерывный край, который встречается с частично закрытым краем. Четыре метки Т-соединения соответствуют закрытию четырех различных типов краев.

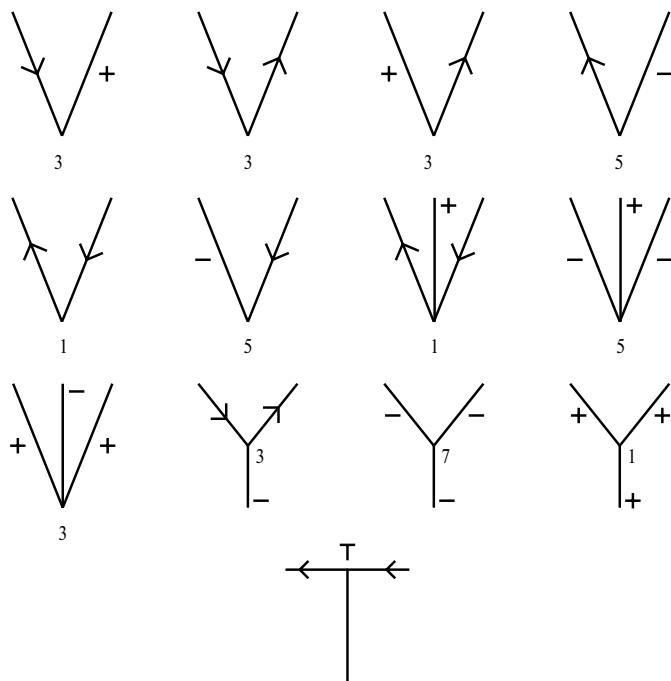


Рис. 24.17. Множество меток Хаффмана-Клоуса

При использовании этого словаря соединений во время поиска разметки для контурного рисунка приходится решать задачу определения того, какие интерпретации соединений являются глобально совместимыми. Соблюдение свойства совместимости обеспечивается путем применения правила, согласно которому каждой линии на рисунке вдоль всей ее длины должна быть присвоена одна и только одна метка. Вальц [1552] предложил алгоритм решения этой задачи (фактически применимый даже для расширенной ее версии с тенями, трещинами и разделено вогнутыми краями), который стал одним из первых приложений метода удовлетворения ограничений в искусственном интеллекте (см. главу 5). В терминологии задач CSP переменными являются соединения, значениями — разметки для этих соединений, а ограничениями служит то, что каждая линия имеет единственную метку. Хотя задача разметки линии для сцен с трехгранными объектами является NP-полной, на практике стандартные алгоритмы CSP показали высокую производительность при их решении.

## 24.5. РАСПОЗНАВАНИЕ ОБЪЕКТОВ

Зрение позволяет нам надежно распознавать людей, животных и неодушевленные объекты. В области искусственного интеллекта или машинного зрения для обозначения всех этих способностей принято использовать термин *распознавание объектов*. К этому относится определение класса конкретных объектов, представленных на изображении (например, лица), а также распознавание самих конкретных объектов (например, лица Билла Клинтона). Ниже перечислены прикладные области, которые стимулируют развитие этого научно-технического направления.

- **Биометрическая идентификация.** Криминальные расследования и контроль доступа на объекты, допускающие присутствие ограниченного круга лиц, требуют наличия возможности однозначно идентифицировать личность людей. Операции снятия отпечатков пальцев, сканирования радужной оболочки глаза и фотографирования лица в фас приводят к получению изображений, которые должны быть сопоставлены с данными, относящимися к конкретным людям.
- **Выборка изображений с учетом их содержимого.** В текстовом документе можно легко найти местонахождение любой строки, например “cat” (кошка), если она там имеется; такую возможность предоставляет любой текстовый редактор. А теперь рассмотрим задачу поиска в изображении подмножества пикселов, которые соответствуют изображению кошки. Если бы система машинного зрения обладала такой способностью, то позволяла бы отвечать на запросы, касающиеся содержимого изображений, такие как “Найдите фотографию, на которой показаны вместе Билл Клинтон и Нельсон Мандела”, “Найдите фотографию конькобежца, который в процессе бега полностью оторвался от льда”, “Найдите фотографию Эйфелевой башни ночью” и т.д., без необходимости вводить ключевые слова, озаглавливающие каждую фотографию в коллекции. По мере того как увеличиваются коллекции фотографий и видеофильмов, задача ввода вручную аннотаций к отдельным объектам из этой коллекции становится все сложнее.
- **Распознавание рукописного текста.** К примерам такого текста относятся подписи, блоки адресов на конвертах, суммы в чеках и введенные пером данные в персональных цифровых ассистентах (Personal Digital Assistant — PDA).

Зрение используется для распознавания не только объектов, но и видов деятельности. Люди способны узнавать знакомую походку (издалека замечая своего друга), выражение лица (улыбку, гримасу), жест (например, просьбу приблизиться), действие (прыжок, танец) и т.д. Исследования по распознаванию видов деятельности все еще находятся на этапе своего становления, поэтому в данном разделе мы сосредоточимся на теме распознавания объектов.

Люди, как правило, легко решают задачу распознавания объектов, но практика показала, что эта задача является сложной для компьютеров. Дело в том, что система машинного зрения должна обладать способностью идентифицировать лицо человека, несмотря на изменения освещенности, позы по отношению к видеокамере и выражения лица. Любое из этих изменений вызывает появление широкого перечня различий в значениях яркости пикселов, поэтому метод, предусматривающий простое сравнение пикселов, вряд ли окажется применимым. Если же требуется обеспечить распознавание экземпляров определенной категории, такой как “автомобили”, то приходится также учитывать различия внутри самой категории. Как оказалось, значительные трудности возникают даже при попытке решить весьма ограниченную проблему распознавания рукописных цифр в поле для почтового кода на конвертах.

Наиболее подходящую инфраструктуру для изучения проблемы распознавания объектов предоставляют такие научные области, как контролируемое обучение или классификация образов. Системе предъявляют положительные примеры изображений (допустим, “лица” — *face*) и отрицательные примеры (допустим, “не лица” — *nonface*) и ставят перед ней задачу определить с помощью обучения функцию, которая позволила бы отнести вновь полученные изображения к одной из двух категорий — *face*, *nonface*. Для достижения этой цели подходят все методы, описанные в главах 18 и 20; в частности, для решения проблем распознавания объектов были применены многослойные персептроны, деревья решений, классификаторы по ближайшим соседним элементам и ядерные машины. Но следует отметить, что задача приспособить эти методы для распознавания объектов — далеко не такая уж простая.

Прежде всего необходимо преодолеть сложности, связанные с сегментацией изображения. Любое изображение, как правило, содержит множество объектов, поэтому необходимо вначале разбить его на подмножества пикселов, соответствующих отдельным объектам. А после разбиения изображения на участки можно ввести данные об этих участках или совокупностях участков в классификатор для определения меток объектов. К сожалению, процесс сегментации “снизу вверх” чреват ошибками, поэтому в качестве альтернативного подхода может быть предусмотрен поиск для определения групп объектов “сверху вниз”. Это означает, что можно проводить поиск подмножества пикселов, которые можно классифицировать как лицо, и в случае успешного выполнения данного этапа результатом становится успешное обнаружение группы! Но подходы, основанные исключительно на поиске “сверху вниз” (или нисходящем поиске), имеют высокую вычислительную сложность, поскольку в них необходимо исследовать окна изображения различных размеров, находящиеся в разных местах, а также сравнивать их все с данными различных гипотез о наличии объектов. В настоящее время такая нисходящая стратегия используется в большинстве практически применяемых систем распознавания объектов, но подобная ситуация может измениться в результате усовершенствования методов поиска “снизу вверх” (восходящего поиска).

Еще одной причиной затруднений является то, что процесс распознавания должен осуществляться надежно, невзирая на изменения освещенности и позы. Люди способны легко распознавать объекты, несмотря на то, что их внешний вид существенно изменяется, даже если судить по данным о значениях яркости пикселов на изображениях этих объектов. Например, мы всегда способны узнать лицо друга при разных условиях освещения или под разными углами зрения. В качестве еще более простого примера рассмотрим задачу распознавания рукописной цифры 6. Люди способны решить такую задачу независимо от изменения размеров и положения такого объекта на изображении, а также несмотря на небольшие изменения угла поворота<sup>3</sup> надписи, изображающей эту цифру.

На данном этапе необходимо сделать одно важное замечание — геометрические трансформации, такие как перенос, масштабирование и поворот, или трансформации яркости изображения, вызванные физическим перемещением источников света, имеют иной характер по сравнению с изменениями внутри категории, например, такими различиями, которыми характеризуются лица разных людей. Очевидно, что единственным способом получения информации о различных типах человеческих лиц или о разных способах написания цифры 4 является обучение. С другой стороны, влияния геометрических и физических трансформаций носят систематический характер, поэтому должна существовать возможность исключить их из рассмотрения на основе продуманного проектирования состава характеристик, используемых для представления обучающих экземпляров.

Практика показала, что одним из весьма эффективных методов обеспечения инвариантности по отношению к геометрическим трансформациям является предварительная обработка рассматриваемого участка изображения и приведение его к стандартной позиции, масштабу и ориентации. Еще один вариант состоит в том, что можно просто игнорировать причинный характер геометрических и физических трансформаций, рассматривая их как дополнительные источники изменчивости изображений, поступающих в классификатор. В таком случае в обучающем множестве необходимо включить экземпляры, соответствующие всем этим вариантам, в расчете на то, что классификатор выявит логическим путем данные о соответствующем множестве трансформаций входных данных, что позволит исключить из рассмотрения указанные причины изменения внешнего вида экземпляров.

Теперь перейдем к описанию конкретных алгоритмов распознавания объектов. Для упрощения сосредоточимся на задаче, постановка которой определена в двухмерной системе координат, — и обучающие, и тестовые примеры заданы в форме двухмерных растровых изображений. Очевидно, что данный подход вполне применим в таких областях, как распознавание рукописного текста. Но даже в случае трехмерных объектов может оказаться эффективным подход, предусматривающий использование способа представления этих объектов с помощью многочисленных двухмерных изображений (рис. 24.18) и классификации новых объектов путем сравнения их с хранимыми изображениями (т.е. с некоторыми другими данными, представляющими те же объекты).

---

<sup>3</sup> Ставить перед собой задачу добиться надежного распознавания при любых углах поворота не нужно и не желательно, поскольку цифру 6 можно повернуть так, что она станет похожей на цифру 9!



Рис. 24.18. Многочисленные изображения двух трехмерных объектов в разных видах

Как было описано в предыдущем разделе, для извлечения из изображения информации о трехмерных объектах в сцене могут использоваться многочисленные признаки. Кроме того, многочисленные признаки лежат в основе распознавания объектов, например, тигра можно узнать, заметив оранжевые и черные цвета на его шкуре, обнаружив на ней полосы или определив форму его тела.

Цвет и текстуру можно представить с использованием гистограмм или эмпирических распределений частот. Получив в качестве образца изображение тигра, можно определить, каково процентное соотношение количества пикселов, относящихся к разным цветам. В дальнейшем, после получения экземпляра с неизвестной классификацией, можно провести сравнение гистограммы его цветов с данными о полученных ранее примерах изображений тигра. Для анализа текстуры рассматриваются гистограммы, полученные в результате свертки изображения с фильтрами, имеющими различные ориентации и масштабы, после чего отыскиваются совпадения.

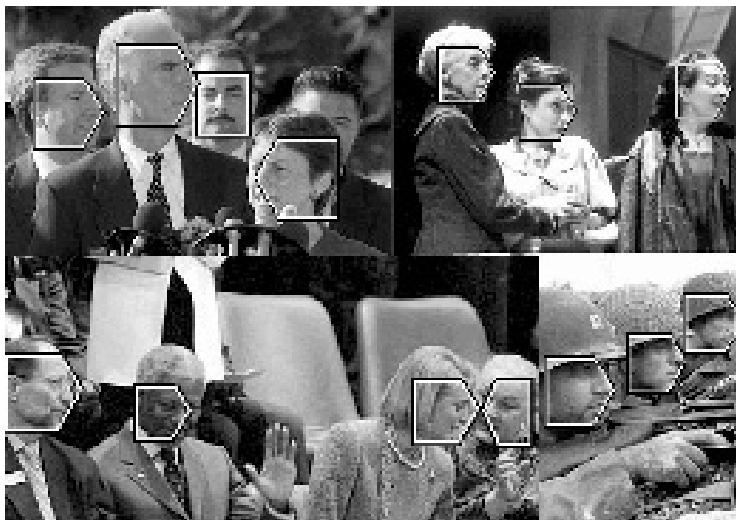
Как оказалось, задача использования формы для распознавания объектов является более сложной. Вообще говоря, существуют два основных подхода: **распознавание с учетом яркости**, в котором непосредственно используются значения яркости пикселов, и **распознавание с учетом характеристик**, в котором предусматривается применение данных о пространственном расположении извлеченных из изображения характеристик, таких как края или ключевые точки. После более подробного описания двух этих подходов мы рассмотрим также проблему **оценки позы**, т.е. проблему определения местонахождения и ориентации объектов в сцене.

### Распознавание с учетом яркости

При таком подходе за основу берется подмножество пикселов изображения, которое соответствует распознаваемому объекту, и определяются данные о характеристиках как данные о самих исходных значениях яркости пикселов. Еще один вариант этого метода состоит в том, что вначале может быть выполнена свертка изображения с различными линейными фильтрами, после чего значения пикселов в результирующих

изображениях рассматриваются как характеристики. Как было показано в разделе 20.7, такой подход оказался очень успешным при решении таких задач, как распознавание рукописных цифр.

Для создания детекторов лиц, позволяющих распознавать лица с помощью баз данных с изображениями, использовался целый ряд статистических методов, включая методы на основе нейронных сетей с необработанными входными данными, представленными в виде характеристик пикселов; деревья решений с характеристиками, определяемыми с помощью различных фильтров полос и краев; а также наивные байесовские модели с характеристиками небольшого волнения (риби). Некоторые результаты применения последнего метода показаны на рис. 24.19.



*Рис. 24.19. Выходные данные алгоритма поиска лиц (любезно предоставлено Генри Шнейдерманом (Henry Schneiderman) и Такео Канаде (Takeo Kanade))*

Одним из недостатков метода, в котором в качестве векторов характеристик используются необработанные данные о пикселях, является большая избыточность, свойственная этому способу представления. Предположим, что рассматриваются два пикселя, расположенные рядом на изображении щеки лица какого-то человека; между ними, скорее всего, будет весьма высокая корреляция, поскольку для них свойственны аналогичное геометрическое расположение, освещенность и т.д. Для сокращения количества размерностей вектора характеристик можно с успехом применять методы уменьшения объема данных, такие как анализ наиболее важных компонентов; использование таких методов обеспечивает распознавание объектов, подобных лицам, с большим быстродействием по сравнению с тем, которое может быть достигнуто с использованием пространства большей размерности.

### Распознавание с учетом характеристик

Вместо применения в качестве характеристик необработанных данных о яркости пикселов можно использовать способы обнаружения и разметки пространственно

локализованных характеристик, таких как участки и края (см. раздел 24.3). Применение краев является целесообразным по двум описанным ниже важным причинам. Одной из них является уменьшение объема данных, связанное с тем, что количество краев намного меньше по сравнению с количеством пикселов изображения. Вторая причина обусловлена возможностью добиться инвариантности освещенности, поскольку края (при наличии подходящего диапазона контрастов) обнаруживаются приблизительно в одних и тех же местах, независимо от точной конфигурации освещеностей. Края представляют собой одномерные характеристики; были также предприняты попытки использовать двухмерные характеристики (участки) и нульмерные характеристики (точки). Обратите внимание на то, как отличаются трактовки пространственного расположения в подходах с учетом яркости и с учетом характеристик. В подходах с учетом яркости эти данные кодируются неявно, как индексы компонентов вектора характеристик, а в подходах с учетом характеристик характеристикой является само местонахождение ( $x, y$ ).

Неотъемлемым свойством любого объекта является инвариантное расположение краев; именно по этой причине люди могут легко интерпретировать контурные рисунки (см. рис. 24.13), даже несмотря на то, что подобные изображения не встречаются в природе! Простейший способ использования этих знаний основан на классификаторе по ближайшим соседним точкам. При этом предварительно вычисляются и сохраняются данные о конфигурациях краев, соответствующие представлениям всех известных объектов. А после получения конфигурации краев, соответствующей неизвестному объекту на изображении, являющимся предметом запроса, можно определить “расстояние” этого объекта от каждого элемента библиотеки хранимых представлений. После этого классификатор по ближайшим соседним точкам выбирает наиболее близкое соответствие.

Для описания понятия расстояния между изображениями было предложено много разных определений. Один из наиболее интересных подходов основан на идее **согласования с учетом деформации**. В своей классической работе *On Growth and Form* [1506] Дарси Томпсон заметил, что близкие, но не идентичные формы часто можно деформировать в подобные друг другу формы с использованием простых координатных преобразований<sup>4</sup>. При таком подходе понятие подобия формы воплощается на практике в виде следующего трехэтапного процесса: во-первых, отыскивается решение задачи соответствия между двумя формами, во-вторых, данные о соответствии используются для определения преобразования, позволяющего сделать эти формы аналогичными, и, в-третьих, вычисляется расстояние между двумя формами как сумма ошибок согласования между соответствующими точками, наряду с термом, в котором измеряется величина выравнивающего преобразования.

Форма представляется с помощью конечного множества точек, полученных в виде выборки, взятой на внутренних или внешних контурах формы. Эти данные могут быть получены как сведения о местонахождениях пикселов краев, обнаруженные детектором краев, и представлены в виде множества  $\{p_1, \dots, p_N\}$  из  $N$  точек. Примеры множеств точек, соответствующих двум формам, приведены на рис. 24.20, *a, б*.

<sup>4</sup> В современной компьютерной графике такой процесс называется **трансформацией**.

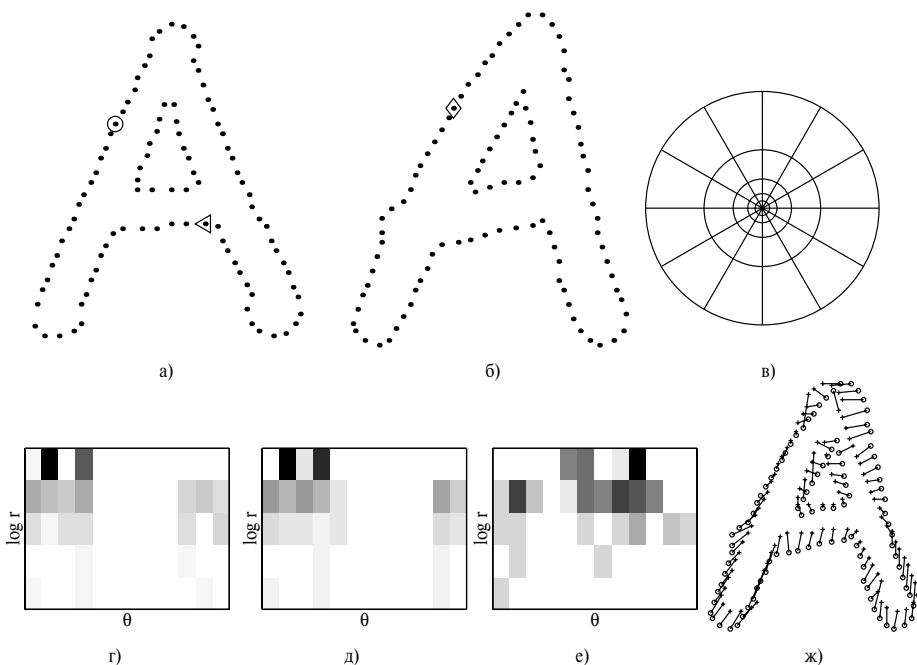


Рис. 24.20. Процедура вычисления и согласования контекстов формы: множества точек краев двух форм (а), (б); схема секторов логарифмической-полярной гистограммы, используемой при вычислении контекстов формы. Используется 5 секторов для  $\log r$  и 12 секторов для  $\theta$  (в); примеры контекстов формы для опорных образцов, отмеченных точками со значками  $\circ$ ,  $\diamond$ ,  $\triangleleft$  на рис. 24.20, а, б. Каждый контекст формы представляет собой логарифмическую-полярную гистограмму координат остальных точек множества, измеренных с использованием опорной точки в качестве начала координат (затемненные ячейки означают, что в данном секторе имеется больше одной точки). Обратите внимание на внешнее подобие контекстов формы знакам  $\circ$  и  $\diamond$ , поскольку эти контексты были вычислены для относительно подобных точек в двух формах. В отличие от этого, контекст формы для знака  $\triangleleft$  существенно отличается (г)–(е); соответствия между рис. 24.20, а, б, обнаружены путем согласования двухходовых графов, с указанием стоимостей преобразования, определяемых на основе расстояния  $\chi^2$  между гистограммами (ж)

Теперь рассмотрим конкретную точку выборки  $p_i$ , наряду с множеством всех векторов, исходящих из этой точки в направлении всех других точек выборки в форме. Эти векторы представляют конфигурацию всей формы относительно рассматриваемой опорной точки. Такое представление лежит в основе следующей идеи: с каждой точкой выборки можно связать дескриптор, или **контекст формы**, который приближенно представляет расположение остальной части формы по отношению к данной точке. Точнее, контекст формы точки  $p_i$  представляет собой приближенную пространственную гистограмму  $h_i$  относительных координат  $p_k - p_i$  остальных  $N-1$  точек  $p_k$ . Для определения сегментов используется логарифмическая-полярная система координат, обеспечивающая то, что дескриптор становится более чувствительным к различиям в ближайших друг к другу пикселях. Пример расположения сегментов показан на рис. 24.20, в.

Обратите внимание на то, что неотъемлемым свойством этого определения контекста формы является его инвариантность к операции переноса, поскольку все изменения выполняются по отношению к точкам в объекте. Для достижения инвари-

антности к операции масштабирования все радиальные расстояния нормализуются путем деления на среднее расстояние между парами точек.

Контексты формы позволяют решить задачу установления соответствия между двумя аналогичными, но не идентичными формами, наподобие тех, которые показаны на рис. 24.20, *a*, *b*. Контексты формы являются разными для различных точек на одной и той же форме  $S$ , тогда как соответствующие (гомологичные) точки на подобных формах  $S$  и  $S'$ , как правило, имеют одинаковые контексты формы. Таким образом, задача поиска соответствующих друг другу точек двух форм преобразована в задачу поиска партнеров, имеющих взаимно подобные контексты формы.

Точнее, рассмотрим точку  $p_i$  на первой форме и точку  $q_j$  на второй форме. Допустим, что  $C_{ij} = C(p_i, q_j)$  обозначает стоимость согласования этих двух точек. Поскольку контексты формы представляют собой распределения, выраженные в виде гистограмм, вполне обоснован подход, предусматривающий использование расстояния  $\chi^2$ , следующим образом:

$$C_{ij} = \frac{1}{2} \sum_{k=1}^K \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

где  $h_i(k)$  и  $h_j(k)$  обозначают  $k$ -й сектор нормализованных гистограмм в точках  $p_i$  и  $q_j$ . Если дано множество стоимостей  $C_{ij}$  согласования между всеми парами точек  $i$  на первой форме и точек  $j$  на второй форме, то может быть принято решение минимизировать общую стоимость согласования с учетом ограничения, что это согласование должно выполняться на основе взаимно-однозначного соответствия. Это — пример задачи **поиска паросочетаний взвешенного двухдолльного графа**, которая может быть решена за время  $O(N^3)$  с использованием так называемого *венгерского алгоритма* (Hungarian algorithm).

Если известны соответствия в точках выборки, то данные о соответствии можно распространить на всю форму, оценивая стоимость согласующего преобразования, которое позволяет отобразить одну форму на другой. Особенно эффективным является подход с использованием регуляризованного тонкостенного сплайна (regularized thin plate spline). После того как формы будут согласованы, задача вычисления оценок подобия становится относительно несложной. Расстояние между двумя формами может быть определено как взвешенная сумма расстояний контекстов форм между соответствующими точками и как энергия изгиба, связанная с тонкостенным сплайном. После получения такой меры расстояния для решения задачи распознавания можно использовать простой классификатор по ближайшим соседним точкам. Превосходная иллюстрация, демонстрирующая высокую эффективность применения этого подхода при классификации рукописных цифр, приведена в главе 20.

## Оценка позы

Интерес представляет не только задача определения того, каковым является некоторый объект, но и задача определения его позы, т.е. его позиции и ориентации по отношению к наблюдателю. Например, при решении проблемы манипулирования объектами на производстве приходится учитывать, что захват робота не может взять объект до тех, пока не будет известна его поза. В случае твердотельных объектов, как

трехмерных, так и двухмерных, эта проблема имеет простое и полностью определенное решение, основанное на **методе выравнивания**, который описан ниже.

В этом методе объект представляется с помощью  $M$  характеристик, или различных точек  $m_1, m_2, \dots, m_M$  в трехмерном пространстве (в качестве таковых могут, допустим, рассматриваться вершины многогранного объекта). Координаты этих точек измеряются в некоторой системе координат, наиболее подходящей для данного объекта. После этого точки подвергаются операции трехмерного вращения  $\mathbf{R}$  с неизвестными параметрами, за которой следует операция переноса на неизвестную величину  $\mathbf{t}$ , а затем выполняется операция проекции, которая приводит к появлению точек характеристик изображения  $p_1, p_2, \dots, p_N$  на плоскости изображения. Вообще говоря,  $N \neq M$ , поскольку некоторые точки модели могут закрывать друг друга, а детектор характеристик может пропустить некоторые характеристики (или выявить ложные характеристики, появление которых обусловлено наличием шума). Такое преобразование для трехмерной модели точек  $m_i$  и соответствующих точек изображения  $p_i$  можно представить следующим образом:

$$p_i = \Pi(\mathbf{R}m_i + \mathbf{t}) = Q(m_i)$$

где  $\mathbf{R}$  — матрица вращения,  $\mathbf{t}$  — вектор переноса;  $\Pi$  — перспективная проекция или одно из ее приближений, такое как масштабируемая ортогональная проекция. Чистым результатом становится трансформация  $Q$ , которая приводит точки модели  $m_i$  в соответствие с точками изображения  $p_i$ . При этом, хотя первоначально трансформация  $Q$  не определена, известно, что (применительно к твердотельным объектам)  $Q$  должна быть одинаковой для всех точек модели.

Задачу определения преобразования  $Q$  можно решить, получив значения трехмерных координат трех точек модели и их двухмерных проекций. В основе этого подхода лежит следующая интуитивная идея: могут быть легко составлены уравнения, связывающие координаты  $p_i$  с координатами  $m_i$ . В этих уравнениях неизвестные величины соответствуют матрице вращения  $\mathbf{R}$  и вектору переноса  $\mathbf{t}$ . Если количество уравнений достаточно велико, то возможность получения решения для  $Q$  становится неоспоримой. Мы не будем приводить здесь доказательство этой гипотезы, а просто сформулируем следующий результат.

Если даны три точки,  $m_1, m_2$  и  $m_3$ , в модели, не лежащие на одной прямой, и их масштабированные ортогональные проекции,  $p_1, p_2$  и  $p_3$ , на плоскости изображения, то существуют две и только две трансформации из системы координат трехмерной модели в систему координат двухмерного изображения.

Эти трансформации связаны друг с другом, поскольку зеркально противоположны относительно плоскости изображения; они могут быть вычислены на основе простого решения в замкнутой форме. Если существует возможность идентифицировать характеристики модели, соответствующие трем характеристикам в изображении, то может быть вычислена  $Q$  — поза объекта. В предыдущем подразделе обсуждался метод определения соответствий с использованием согласования контекстов формы. Если же объект имеет четко определенные углы или другие заметные точки, то становится доступным еще более простой метод. Идея его состоит в том, что необходимо повторно формировать и проверять соответствия. Мы должны выдвинуть первоначальную гипотезу о соответствии тройки точек изображения тройке точек модели и использовать функцию *Find-Transform* для формирования гипотезы  $Q$ .

Если принятное предположение о соответствии было правильным, то трансформация  $Q$  является правильной и после ее применения к оставшимся точкам модели приводит к получению предсказания координат точек изображения. Если принятное предположение было неправильным, то трансформация  $Q$  также является неправильной и после ее применения к оставшимся точкам модели не позволяет предсказывать координаты точек изображения.

Описанный выше подход лежит в основе алгоритма Align, приведенного в листинге 24.1. Этот алгоритм находит позу для данной конкретной модели или возвращает индикатор неудачи. Временная сложность данного алгоритма в худшем случае пропорциональна количеству сочетаний троек точек модели и троек точек изображения, или

$$\binom{N}{3} \binom{M}{3},$$

умноженному на стоимость проверки каждого сочетания. Стоимость проверки пропорциональна  $M \log N$ , поскольку необходимо предсказывать позицию изображения для каждой из  $M$  точек модели и находить расстояние до ближайшей точки изображения, что требует выполнения  $\log N$  операций, если точки изображения представлены с помощью некоторой подходящей структуры данных. Поэтому в наихудшем случае сложность алгоритма выравнивания определяется значением  $O(M^4 N^3 \log N)$ , где  $M$  и  $N$  — количество точек модели и изображения соответственно. Методы, основанные на принципе кластеризации поз в сочетании со средствами рандомизации, позволяют уменьшить сложность до  $O(MN^3)$ . Результаты применения этого алгоритма к изображению степлера показаны на рис. 24.21.

#### Листинг 24.1. Неформальное описание алгоритма выравнивания

---

```

function Align(image, model) returns решение solution или индикатор
    неудачи failure
    inputs: image, список координат характеристик изображения
            model, список координат характеристик модели

    for each (p1, p2, p3) in Triplets(image) do
        for each (m1, m2, m3) in Triplets(model) do
            Q  $\leftarrow$  Find-Transform((p1, p2, p3), (m1, m2, m3))
            if проекция, соответствующая гипотезе Q,
                позволяет распознать изображение then
                return Q
    return failure

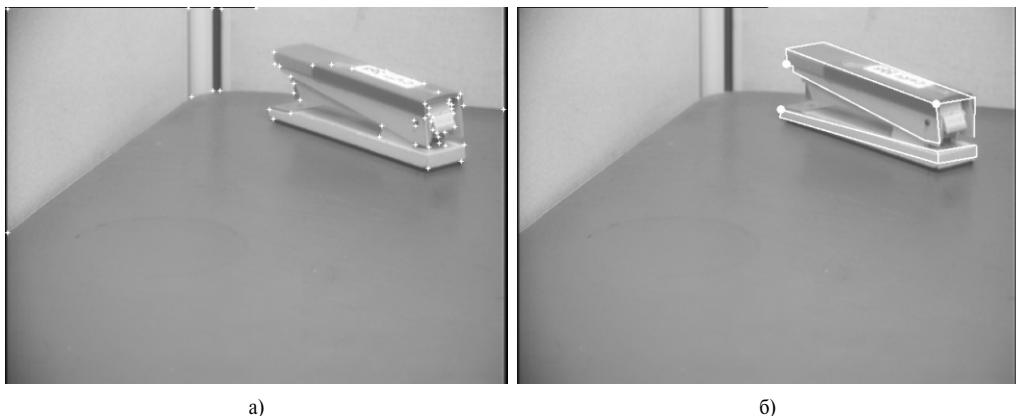
```

---

## 24.6. ИСПОЛЬЗОВАНИЕ СИСТЕМЫ МАШИННОГО ЗРЕНИЯ ДЛЯ МАНИПУЛИРОВАНИЯ И ПЕРЕДВИЖЕНИЯ

Одно из наиболее важных направлений использования систем машинного зрения состоит в получении информации как для манипулирования объектами (определения их местоположения, захвата, изменения их положения в пространстве и т.д.), так и для передвижения без столкновений с препятствиями. Способность использовать зрение для этих целей присуща системам зрения даже самых примитив-

ных животных. Во многих случаях по своему устройству такая система зрения состоит из минимально необходимого набора компонентов; под этим подразумевается, что она извлекает из доступного светового поля только такую информацию, которая требуется животному для организации своего поведения. Вполне возможно, что системы зрения наиболее высокоразвитых животных стали результатом эволюции, которая началась с появления на одном конце тела у самых ранних, примитивных организмов светочувствительного пятна, с помощью которого они устремлялись к свету (или прятались от него). Как было описано в разделе 24.4, в нервной системе муhi существует очень простая система распознавания оптического потока, позволяющая муhi садиться на стены. В классическом исследовании *What the Frog's Eye Tells the Frog's Brain* [914] сделано следующее замечание в отношении лягушки: “Она умерла бы с голоду, окруженнная пищей, если бы эта пища не двигалась. Лягушка выбирает пищу только после определения ее размеров и движения”.



*Рис. 24.21. Результаты применения алгоритма выравнивания к изображению степлера; углы, найденные на изображении степлера (а); гипотетическая реконструкция, наложенная на первоначальное изображение (любезно предоставлено Кларком Олсоном (Clark Olson)) (б)*

Системы машинного зрения используются в “организмах”, называемых роботами. Рассмотрим особую разновидность робота — автоматизированное транспортное средство, движущееся по шоссе (рис. 24.22). Вначале проанализируем стоящие перед нами задачи, затем определим, какие алгоритмы машинного зрения позволят нам получить информацию, необходимую для успешного выполнения этих задач. Ниже перечислены задачи, с которыми сталкивается водитель.

1. Управление движением в поперечном направлении. Обеспечение того, чтобы транспортное средство надежно придерживалось своей полосы движения или плавно переходило на другую полосу движения в случае необходимости.
2. Управление движением в продольном направлении. Обеспечение того, чтобы постоянно соблюдалась безопасная дистанция до транспортного средства, идущего впереди.
3. Предотвращение столкновений с препятствиями. Слежение за транспортными средствами на соседних полосах движения и подготовка к маневрам, необходимым для предотвращения столкновения, если водитель одного из них решит перейти на другую полосу движения.

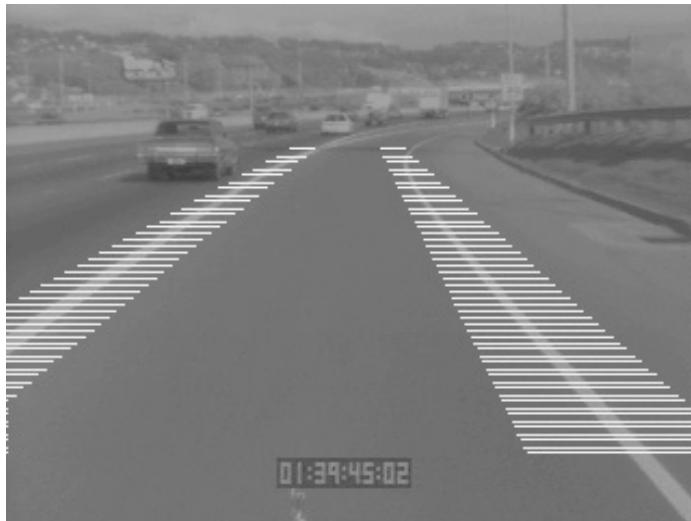


Рис. 24.22. Изображение дороги, снятое видеокамерой, которая расположена внутри автомобиля. Горизонтальными белыми полосами обозначены окна поиска, в пределах которых контроллер отыскивает маркировку полосы движения. Низкое качество изображения является вполне типичным для черно-белого видеосигнала с низким разрешением

Перед водителем стоит проблема — определить и осуществить подходящие действия по изменению направления движения, ускорению и торможению, позволяющие наилучшим образом выполнить стоящие перед ним задачи.

Что касается управления движением в поперечном направлении, то для этого необходимо постоянно обновлять данные о положении и ориентации автомобиля относительно его полосы движения. Применительно к изображению, показанному на рис. 24.22, для поиска краев, соответствующих сегментам маркировки полосы движения, можно использовать алгоритмы распознавания краев. После этого с данными элементами представления краев можно согласовать гладкие кривые. Параметры этих кривых несут информацию о поперечном положении автомобиля, направлении, в котором он движется относительно своей полосы движения, и о кривизне самой полосы движения. Эта информация, наряду с информацией о динамике автомобиля, заключает в себе все необходимое для системы рулевого управления. Следует также отметить, что от одного видеокадра к другому происходит лишь небольшое изменение в положении проекции полосы движения на изображении, поэтому уже известно, где искать на изображении маркировку полосы движения; например, на данном рисунке достаточно рассмотреть только те участки, которые обозначены параллельными белыми полосками.

А что касается управления движением в продольном направлении, то необходимо знать расстояния до идущих впереди транспортных средств. Для получения такой информации могут использоваться бинокулярные стереооданные или оптический поток. Оба эти подхода могут быть упрощены с использованием ограничений проблемной области, определяемых тем фактом, что вождение происходит на плоской поверхности. В настоящее время автомобили, действующие под управлением систем

машинного зрения, в которых используются эти методы, показали свою способность двигаться в течение продолжительных периодов времени на максимальных скоростях, разрешенных на автомагистралях.

Приведенный выше пример решения проблемы вождения позволяет очень четко подчеркнуть одну мысль: *для решения конкретной задачи нет необходимости извлекать из изображения всю информацию, которая может быть в принципе получена с его помощью*. Не требуется восстанавливать точную форму каждого встречного или попутного автомобиля, решать задачу определения формы на основании текстуры для поверхности травы, растущей вдоль автомагистрали, и т.д. Потребности данной задачи определяют необходимость в получении лишь информации определенных видов, и поэтому можно добиться значительного повышения скорости вычислений и надежности, восстанавливая только эту информацию и в полной мере применяя ограничения проблемной области. Наша цель при обсуждении общих подходов, представленных в предыдущем разделе, состояла в демонстрации того, что они формируют общую теорию, которую можно специализировать в интересах решения конкретных задач.

## 24.7. РЕЗЮМЕ

Хотя на первый взгляд кажется, что люди осуществляют действия по восприятию без каких-либо усилий, для обеспечения восприятия требуется большой объем сложных вычислений. Задача зрения состоит в извлечении информации, необходимой для решения таких задач, как манипулирование, навигация и распознавание объектов.

- Геометрические и физические аспекты процесса **формирования изображения** глубоко изучены. Если дано описание трехмерной сцены, можно легко сформировать ее изображение из любой произвольной позиции видеокамеры (это — задача компьютерной графики). Задача организации обратного процесса, в котором происходит переход от изображения к описанию сцены, является более сложной.
- Для извлечения визуальной информации, необходимой для решения задач манипулирования, навигации и распознавания, необходимо создавать промежуточные представления. В ранних алгоритмах **обработки изображения** для систем машинного зрения предусматривалось извлечение из изображения таких примитивных характеристик, как края и участки.
- В каждом изображении имеется целый ряд признаков, позволяющих получить информацию о конфигурации рассматриваемой трехмерной сцены: движение, стереоданные, текстура, затенение и контуры. Выделение каждого из этих признаков основано на исходных допущениях о физических сценах, позволяющих добиваться почти полностью непротиворечивых интерпретаций.
- Задача распознавания объектов в своей полной постановке является весьма сложной. В данной главе рассматривались подходы к решению этой задачи с учетом яркости и характеристик. Кроме того, в настоящей главе приведен простой алгоритм оценки позы. Существуют и другие возможности.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Упорные попытки понять, как функционирует зрение человека, предпринимались с самых древних времен. Евклид (около 300 г. до н.э.) в своих трудах писал о естественной перспективе — об отображении, которое связывает с каждой точкой  $P$  в трехмерном мире направление луча  $OP$ , соединяющего центр проекции  $O$  с точкой  $P$ . Он также был хорошо знаком с понятием параллакса движения. Следующий значительный этап развития математической трактовки перспективной проекции, на этот раз в контексте проекции на плоские поверхности, наступил в XV веке в Италии, в период Возрождения. Создателем первых рисунков, основанных на геометрически правильной проекции трехмерной сцены, принято считать Брунеллески (1413 год). В 1435 году Альберти составил свод правил построения перспективной проекции, ставший источником вдохновения для многих поколений художников, чьи художественные достижения восхищают нас и поныне. Особенно весомый вклад в развитие *науки о перспективе* (как она называлась в те времена) внесли Леонардо да Винчи и Альбрехт Дюрер. Составленные Леонардо в конце XV столетия описания игры света и тени (светотени), теневых и полутеневых областей затенения, а также воздушной перспективы до сих пор не потеряли своего значения [790].

Хотя знаниями о перспективе владели еще древние греки, в их воззрениях присутствовала забавная путаница, поскольку они неправильно понимали, какую роль играют глаза в процессе зрения. Аристотель считал, что глаза — это устройства, испускающие лучи, что соответствует современным представлениям о работе лазерных дальномеров. Этим ошибочным взглядам положили конец труды арабских ученых X столетия, в частности Альхазена. В дальнейшем началась разработка камер-обскур различных видов. На первых порах они представляли собой комнаты (камера-обскура по латыни — “темная комната”), в которые свет попадал через малое отверстие в одной стене, а на противоположной стене создавалось изображение сцены, происходящей наружу. Безусловно, во всех этих камерах изображение было перевернутым, что вызывало невероятное смущение современников. Ведь если глаз рассматривать как аналогичный такому устройству формирования изображения, как камера-обскура, то почему же мы видим предметы такими, каковы они на самом деле? Эта загадка не давала покоя величайшим умам той эпохи (включая Леонардо). Окончательно решить эту проблему удалось лишь благодаря работам Кеплера и Декарта. Декарт поместил препарат глаза, с задней стенки которого была удалена непрозрачная оболочка, в отверстие оконного ставня. В результате было получено перевернутое изображение, сформировавшееся на куске бумаги, заменившем сетчатку. Хотя изображение на сетчатке глаза действительно перевернуто, такая ситуация не вызывает проблемы, поскольку мозг интерпретирует полученное изображение правильно. Говоря современным языком, для этого достаточно обеспечить правильный доступ к структуре данных.

Очередные крупные успехи в изучении зрения были достигнуты в XIX веке. Благодаря трудам Гельмгольца и Вундта, описанным в главе 1, методика проведения психофизических экспериментов стала строгой научной дисциплиной. А труды Юнга, Максвелла и Гельмгольца привели к созданию трехкомпонентной теории цветоощущения. Стереоскоп, изобретенный Витстоуном [1582], позволил продемонстрировать, что люди получают возможность определять глубину изображения,

если на левый и правый глаз поступают немного разные картинки. После того как стало известно о создании стереоскопа, этот прибор быстро завоевал популярность в гостиных и салонах по всей Европе. Возникла новая научная область — *фотограмметрия*, основанная на принципиально важном понятии бинокулярных стереоданных, согласно которому два изображения сцены, снятые немного с разных точек зрения, несут достаточную информацию для получения трехмерной реконструкции сцены. В дальнейшем были получены важные математические результаты; например, Круппа [861] доказал, что если даны два изображения пяти различных точек одного и того же объекта, то можно реконструировать данные о повороте и переносе камеры с одной позиции в другую, а также о глубине сцены (с точностью до коэффициента масштабирования). Хотя геометрия стереоскопического зрения была известна уже давно, не было ясно, как решают задачу фотограмметрии люди, автоматически согласующие соответствующие точки изображений. Удивительные способности людей решать проблему соответствия были продемонстрированы Юлешем [755], который изобрел стереограмму, состоящую из случайно выбранных точек. На решение проблемы соответствия как в машинном зрении, так и в фотограмметрии в 1970-х и в 1980-х годах были потрачены значительные усилия.

Вторая половина XIX столетия была основным периодом становления области психофизических исследований человеческого зрения. В первой половине XX столетия наиболее значительные результаты исследований в области зрения были получены представителями школы гештальт-психологии, возглавляемой Максом Вертхаймером. Эти ученые были проводниками взглядов, что основными единицами восприятия должны быть законченные формы, а не их компоненты (такие как края), и выдвинули лозунг: “Целое не равно сумме его частей”.

Период исследований после Второй мировой войны характеризуется новым всплеском активности. Наиболее значительной была работа Дж.Дж. Гибсона [551], [552], который подчеркнул важность понятий оптического потока, а также градиентов текстуры в оценке таких переменных описания внешней среды, как поворот и наклон поверхности. Гибсон еще раз подчеркнул значимость стимулов и их разнообразия. Например, в [553] указано, что поле оптического потока всегда содержит достаточно информации для определения самодвижения наблюдателя по отношению к его среде. В сообществе специалистов по системам компьютерного зрения основные работы в этой области и в (математически эквивалентной) области выявления структуры по данным о движении проводились главным образом в 1980-х и в 1990-х годах. Наиболее яркими проявлениями этой деятельности стали оригинальные работы [815], [945] и [1526]. Возникавшая на первых порах озабоченность в отношении стабильности структуры, выявленной на основании данных о движении, была полностью развеяна благодаря работе Томази и Канаде [1511], которые показали, что форма может быть восстановлена абсолютно точно благодаря использованию многочисленных кадров и получаемой в результате этого широкой базисной линии.

В [230] описано удивительное устройство системы зрения мухи и показано, что это насекомое обладает остротой временного визуального восприятия, в десять раз лучшей по сравнению с человеком. Это означает, что муха способна смотреть фильм, воспроизводимый с частотой до 300 кадров в секунду, различая при этом отдельные кадры.

Принципиально важным нововведением, представленным в исследованиях, которые проводились в 1990-х годах, было выявление с помощью обучения проектив-

ной структуры по данным о движении. Как показано в [452], при таком подходе не требуется калибровка видеокамеры. Это открытие тесно связано с работами, послужившими основой для использования геометрических инвариантов при распознавании объектов, обзор которых приведен в [1104], и с работами по разработке аффинной структуры по данным о движении [816]. В 1990-х годах анализ движения нашел много новых областей применения благодаря значительному увеличению быстродействия и объема памяти компьютеров, а также широкому распространению цифровой видеоаппаратуры. Особенно важное применение нашли методы создания геометрических моделей сцен реального мира, которые предназначены для формирования изображений с помощью средств компьютерной графики; эти работы привели к созданию алгоритмов реконструкции наподобие тех, которые представлены в [364]. В [454] и [626] приведено исчерпывающее описание геометрии множественных представлений.

В области компьютеризированных систем зрения наиболее важными основополагающими работами по логическому выводу формы на основании текстуры были [60] и [1461]. Они были посвящены описанию плоских поверхностей, а для криволинейных поверхностей результаты исчерпывающего анализа приведены в [518] и [973].

В сообществе специалистов в области компьютерных систем зрения проблема логического вывода формы из данных о затенении была впервые исследована Бертольдом Хорном [676]. В [678] представлен исчерпывающий обзор основных статей в этой области. В указанном научном направлении было принято принимать целый ряд упрощающих допущений, из которых наиболее важным было игнорирование влияния взаимного освещения. Важность проблемы взаимного освещения была впервые осознана в сообществе специалистов по компьютерной графике, которые стремились точно разрабатывать модели трассировки лучей и диффузного отражения, чтобы учсть наличие взаимного освещения. С критикой основных теоретических и эмпирических подходов в этой области можно ознакомиться в [484].

В области логического вывода информации о форме по данным о контурах самый первый, решающий вклад был сделан Хаффменом [702] и Клоувсом [271], после чего Маккуорт [966] и Сугихара [1473] провели до конца анализ методов, применимых к многогранным объектам. Малик [971] разработал схему разметки для кусочно-гладких криволинейных объектов. В [801] показано, что задача разметки линий для трехгранных сцен является NP-полной.

Для правильной трактовки визуальных эффектов, возникающих в проекциях гладких криволинейных объектов, требуется совместное использование дифференциальной геометрии и теории особенностей. Наилучшим исследованием на эту тему является книга Кёндеринка *Solid Shape* [814].

Оригинальной работой по распознаванию трехмерных объектов явились тезисы Роберта [1294], опубликованные в Массачусетском технологическом институте (Massachusetts Institute of Technologies — MIT). Эту работу многие считают первыми тезисами докторской диссертации по машинному зрению; в ней впервые представлено несколько ключевых идей, в том числе касающихся обнаружения краев и согласования на основе моделей. Метод обнаружения краев Кэнни был представлен в [218]. Идея выравнивания, также впервые выдвинутая Робертом, снова вышла на передний план в 1980-х годах после опубликования работ [711] и [950]. Значительное повышение эффективности методов оценки позы путем выравнивания было достигнуто Олсоном [1156]. Еще одним важным направлением исследований в области распознавания

трехмерных объектов явился подход, основанный на идеи описания форм в терминах объемных примитивов на основе **обобщенных цилиндров**, который был предложен Томом Бинфордом [128] и нашел особенно широкое распространение.

Исследования в области машинного зрения, посвященные распознаванию объектов, в основном сосредоточились на проблемах, возникающих в результате получения проекции трехмерных объектов в виде двухмерных изображений, а в сообществе специалистов по распознаванию образов существовала другая традиция, в которой эта задача рассматривалась как относящаяся к области классификации образов. Этих специалистов в основным интересовали примеры, относящиеся к таким проблемным областям, как оптическое распознавание символов и распознавание рукописных почтовых кодов, в которых основные усилия были направлены на изучение характеристик типичных вариаций искомого класса объектов и отделение этих объектов от объектов других классов. Сравнение таких подходов приведено в [904]. К другим работам по распознаванию объектов относятся исследования по распознаванию лиц [1422] и [1543]. В [98] описан подход на основе контекста формы. В [395] впервые показаны результаты разработки автомобиля с визуальным управлением для автоматического вождения по автомагистралям на высоких скоростях; в [1224] показаны результаты достижения аналогичной производительности с использованием подхода на основе нейронной сети.

Наилучшее и самое полное описание человеческого зрения можно найти в книге Стивена Палмера *Vision Science: Photons to Phenomenology* [1167]; а книги Дэвида Хабела *Eye, Brain and Vision* [700] и Ирвина Рока *Perception* [1300] представляют собой краткие введения, в основном посвященные соответственно нейрофизиологии и восприятию.

В настоящее время наиболее всесторонним учебником для специалистов по машинному зрению является книга Дэвида Форсита (David Forsyth) и Джин Понсе (Jean Ponce) *Computer Vision: A Modern Approach*. Значительно более краткие описания можно найти в [1111] и [1513]. Интерес представляют также два изданных немного раньше, но все еще значимых учебника, в каждом из которых рассматривается ряд специальных тем: *Robot Vision* [677] и *Three-Dimensional Computer Vision* [453]. Важную роль в объединении усилий специалистов по машинному зрению и специалистов по более традиционным областям биологического зрения (психофизике и нейробиологии) в свое время сыграла книга Дэвида Марпа *Vision* [986]. Двумя основными журналами по машинному зрению являются *IEEE Transactions on Pattern Analysis and Machine Intelligence* и *International Journal of Computer Vision*. К числу конференций по машинному зрению относятся *ICCV* (International Conference on Computer Vision), *CVPR* (Computer Vision and Pattern Recognition) и *ECCV* (European Conference on Computer Vision).

## УПРАЖНЕНИЯ

---

- 24.1.** В тени дерева с плотной, густой кроной можно обнаружить множество пятен света. На удивление, все эти пятна кажутся круглыми. С чем это связано? Ведь в конечном итоге просветы между листьями, через которые проникают лучи солнца, вряд ли имеют круглую форму.

- 24.2.** Нанесите разметку на контурный рисунок, приведенный на рис. 24.23, приняв предположение, что все наружные края размечены как закрывающие и что все вершины являются трехгранными. Выполните эту задачу с помощью алгоритма перебора с возвратами, который проверяет вершины в последовательности  $A, B, C$  и  $D$ , выбирая на каждом этапе вариант, совместимый с размеченными ранее соединениями и краями. После этого попробуйте применить последовательность вершин  $B, D, A$  и  $C$ .

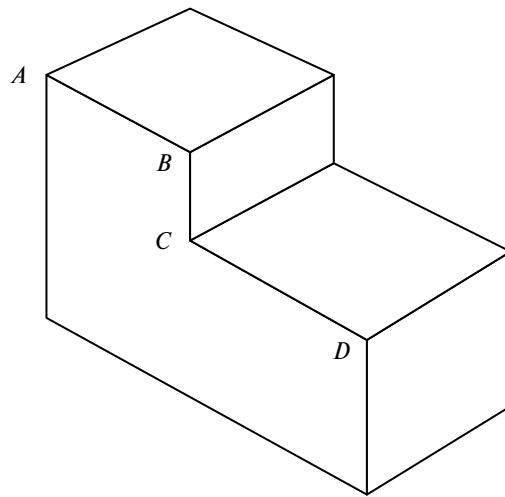


Рис. 24.23. Рисунок, подлежащий разметке, в котором все вершины являются трехгранными

- 24.3.** Рассмотрим цилиндр бесконечной длины с радиусом  $x$ , ось которого направлена вдоль оси  $y$ . Цилиндр имеет ламбертову поверхность и рассматривается с помощью видеокамеры, направленной вдоль положительной оси  $z$ . Что вы можете рассчитывать увидеть на этом изображении, если цилиндр освещается точечным источником света, находящимся на бесконечно большом расстоянии со стороны положительной полуоси  $x$ ? Объясните ваш ответ, нарисовав контуры равной яркости на спроектированном изображении. Являются ли контуры равной яркости расположеными через равномерные интервалы?
- 24.4.** Края на изображении могут соответствовать самым разнообразным визуальным эффектам, возникающим в сцене. Рассмотрите обложку данной книги и примите предположение, что это — картина реальной трехмерной сцены. Определите на этом изображении десять краев с различной яркостью и для каждого из них укажите, соответствует ли оно сосредоточенной неоднородности:
- по глубине;
  - по нормали к поверхности;
  - по отражательной способности;
  - по освещенности.

- 24.5.** Покажите, что операция свертки с заданной функцией  $f$  является коммутативной по отношению к операции дифференцирования; иными словами, покажите, что  $(f*g)' = f*(g)'$ .
- 24.6.** Рассматривается вопрос о возможности применения некоторой стереоскопической системы для составления карты местности. Она состоит из двух видеокамер CCD, в каждой из которых имеется  $512 \times 512$  пикселов на квадратном датчике площадью  $10 \times 10$  см. Применяемые линзы имеют фокусное расстояние 16 см, где фокус зафиксирован в бесконечности. Для соответствующих точек с координатами  $(u_1, v_1)$  на левом изображении и  $(u_2, v_2)$  на правом изображении верно, что  $v_1=v_2$ , поскольку оси  $x$  двух плоскостей изображения параллельны эпиполярным линиям. Оптические оси этих двух видеокамер являются параллельными. Базисная линия между камерами равна 1 м.
- Если наименьшая дальность, которая должна быть измерена, равняется 16 м, то каково наибольшее рассогласование (в пикселях), которое может при этом возникнуть?
  - Какова разрешающая способность по дальности на расстоянии 16 м, которая обусловлена наличием интервала между пикселями?
  - Какая дальность соответствует рассогласованию в один пикセル?
- 24.7.** Предположим, что нужно применить алгоритм выравнивания в промышленной установке, в которой по ремню конвейера движутся плоские детали и фотографируются видеокамерой, находящейся вертикально над ремнем конвейера. Поза детали задается тремя переменными: одна из них определяет поворот, а две другие — положение относительно двух горизонтальных осей. Тем самым задача упрощается, а для функции Find-Transform требуется, чтобы позу определяли только две пары соответствующих характеристик изображения и модели. Определите сложность этой процедуры выравнивания в наихудшем случае.
- 24.8.** (*Любезно предоставлено Pietro Peronой (Pietro Perona).*) На рис. 24.24 показаны две видеокамеры в точках  $X$  и  $Y$ , с помощью которых ведется наблюдение за сценой. Нарисуйте изображение, поступающее на каждую видеокамеру, приняв предположение, что все обозначенные точки находятся на одной и той же горизонтальной плоскости. Можно ли с помощью этих двух изображений сделать заключение об относительных расстояниях точек  $A$ ,  $B$ ,  $C$ ,  $D$  и  $E$  от базисной линии видеокамер? На чем должно быть основано это заключение?
- 24.9.** Какие из приведенных ниже утверждений являются истинными и какие ложными?
- Обнаружение соответствующих друг другу точек в стереоскопических изображениях — самая простая стадия процесса стереоскопического поиска глубины.
  - Извлечение формы из текстуры можно выполнить, проектируя на сцену сетку световых полос.
  - Схема разметки Хаффмена–Клоувса предназначена для использования с любыми многогранными объектами.

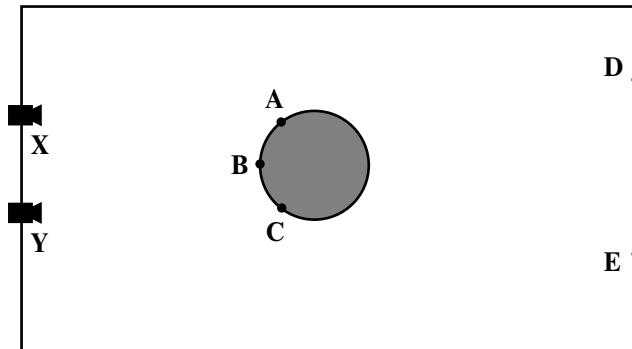


Рис. 24.24. Вид сверху системы машинного зрения с двумя видеокамерами, в которой ведется наблюдение за бутылью и стоящей сзади ее стеной

- г) На контурных рисунках криволинейных объектов метка линии по мере прохождения от одного конца линии к другому может изменяться.
  - д) При использовании стереоскопических изображений одной и той же сцены большая точность вычисления глубины достигается, если две камеры расположены дальше друг от друга.
  - е) Проекциями линий равной длины в сцене всегда становятся линии равной длины в изображении.
  - ж) Прямые линии в изображении обязательно соответствуют прямым линиям в сцене.
- 24.10.** Фрагмент видеоизображения, приведенный на рис. 24.22, снят из автомобиля, находящегося на полосе движения, которая предназначена для выезда с автострады. На полосе движения, расположенной непосредственно слева, видны два автомобиля. На каком основании наблюдатель мог бы заключить, что один из них ближе к нему, чем другой?

# 25 РОБОТОТЕХНИКА

*В этой главе описано, как оснастить агентов физическими исполнительными механизмами, чтобы они могли немного пошалить.*

## 25.1. ВВЕДЕНИЕ

↗ **Роботы** — это физические агенты, которые выполняют поставленные перед ними задачи, проводя манипуляции в физическом мире. Для этой цели роботов оснашают ↗ **исполнительными механизмами**, такими как ноги, колеса, шарниры и захваты. Исполнительные механизмы имеют единственное назначение — прилагать физические усилия к среде<sup>1</sup>. Кроме того, роботов оснашают ↗ **датчиками**, которые позволяют им воспринимать данные об окружающей их среде. В современных роботах применяются различные виды датчиков, включая те, что предназначены для измерения характеристик среды (например, видеокамеры и ультразвуковые дальномеры), и те, которые измеряют характеристики движения самого робота (например, гироскопы и акселерометры).

Большинство современных роботов относится к одной из трех основных категорий. ↗ **Роботы-манипуляторы**, или роботы-руки, физически привязаны к своему рабочему месту, например на заводском сборочном конвейере или на борту Международной космической станции. В движении манипулятора обычно участвует вся цепочка управляемых шарниров, что позволяет таким роботам устанавливать свои исполнительные механизмы в любую позицию в пределах своего рабочего пространства. Манипуляторы относятся к типу наиболее распространенных промышленных роботов, поскольку во всем мире установлено свыше миллиона таких устройств. Некоторые мобильные манипуляторы используются в больницах в качестве ассистентов хирургов. Без робототехнических манипуляторов в наши дни не смогут продолжать свою производственную деятельность большинство автомобильных заводов, а некоторые манипуляторы использовались даже для создания оригинальных художественных произведений.

---

<sup>1</sup> В главе 2 речь шла, скорее, об исполнительных системах (actuator), а не об исполнительных механизмах (effector). Исполнительной системой называется система управления, которая передает команды исполнительному механизму, а исполнительным механизмом называется само физическое устройство.

Ко второй категории относятся **мобильные роботы**. Роботы такого типа передвигаются в пределах своей среды с использованием колес, ног или аналогичных механизмов. Они нашли свое применение при доставке обедов в больницах, при перемещении контейнеров в грузовых доках, а также при выполнении аналогичных задач. В этой книге уже встречался один пример мобильного робота — **автоматическое наземное транспортное средство** (Unmanned Land Vehicle — ULV) NavLab, способное автономно передвигаться по автомагистралям в режиме самовождения. К другим типам мобильных роботов относятся **автоматическое воздушное транспортное средство** (Unmanned Air Vehicle — UAV), обычно используемое для воздушного наблюдения, химической обработки земельных участков и военных операций, **автономное подводное транспортное средство** (Autonomous Underwater Vehicle — AUV), используемое в глубоководных морских исследованиях, и **планетоход**, такой как робот Sojourner, показанный на рис. 25.1, а.

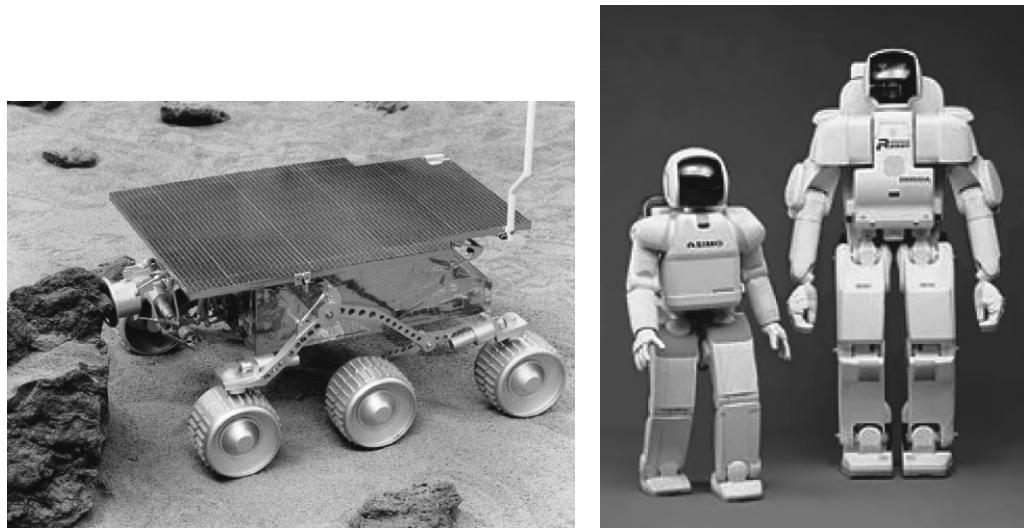


Рис. 25.1. Фотографии широко известных роботов: движущийся робот Sojourner агентства NASA, который исследовал поверхность Марса в июле 1997 года (а); роботы-гуманоиды P3 и Asimo компании Honda (б)

К третьему типу относятся гибридные устройства — мобильные роботы, оборудованные манипуляторами. В их число входят **роботы-гуманоиды**, которые по своей физической конструкции напоминают человеческое тело. Два таких робота-гуманоида показаны на рис. 25.1, б; оба они изготовлены в японской корпорации Honda. Гибридные роботы способны распространить действие своих исполнительных элементов на более обширную рабочую область по сравнению с прикрепленными к одному месту манипуляторами, но вынуждены выполнять стоящие перед ними задачи с большими усилиями, поскольку не имеют такой жесткой опоры, которую предоставляет узел крепления манипулятора.

К сфере робототехники относятся также протезные устройства (искусственные конечности, ушные и глазные протезы для людей), интеллектуальные системы жиз-

необеспечения (например, целые дома, оборудованные датчиками и исполнительными механизмами), а также многотельные системы, в которых робототехнические действия осуществляются с использованием целых полчищ небольших роботов, объединяющих свои усилия.

Реальным роботам обычно приходится действовать в условиях среды, которая является частично наблюдаемой, стохастической, динамической и непрерывной. Некоторые варианты среды обитания роботов (но не все) являются также последовательными и мультиагентными. Частичная наблюдаемость и стохастичность обусловлены тем, что роботу приходится сталкиваться с большим, сложным миром. Робот не может заглянуть за каждый угол, а команды на выполнение движений осуществляются не с полной определенностью из-за проскальзывания приводных механизмов, трения и т.д. Кроме того, реальный мир упорно отказывается действовать быстрее, чем в реальном времени. В моделируемой среде предоставляется возможность использовать простые алгоритмы (такие как алгоритм **Q-обучения**, описанный в главе 21), чтобы определить с помощью обучения необходимые параметры, осуществляя миллионы попыток в течение всего лишь нескольких часов процессорного времени, а в реальной среде для выполнения всех этих попыток могут потребоваться годы. Кроме того, реальные аварии, в отличие от моделируемых, действительно наносят ущерб. В применяемые на практике робототехнические системы необходимо вносить априорные знания о роботе, о его физической среде и задачах, которые он должен выполнять для того, чтобы быстро пройти обучение и действовать безопасно.

## 25.2. АППАРАТНОЕ ОБЕСПЕЧЕНИЕ РОБОТОВ

До сих пор в этой книге предполагалось, что конструкция компонентов архитектуры агентов (датчиков, исполнительных механизмов и процессоров) уже определена и осталось лишь заняться разработкой программы агента. Но успехи в создании реальных роботов не в меньшей степени зависят от того, насколько удачно будут спроектированы датчики и исполнительные механизмы, подходящие для выполнения поставленной задачи.

### Датчики

Датчики — это не что иное, как интерфейс между роботами и той средой, в которой они действуют, обеспечивающий передачу результатов восприятия. **Пассивные датчики**, такие как видеокамеры, в полном смысле этого слова выполняют функции наблюдателя за средой — они перехватывают сигналы, создаваемые другими источниками сигналов в среде. **Активные датчики**, такие как локаторы, посыпают энергию в среду. Их действие основано на том, что часть излучаемой энергии отражается и снова поступает в датчик. Как правило, активные датчики позволяют получить больше информации, чем пассивные, но за счет увеличения потребления энергии от источника питания; еще одним их недостатком является то, что при одновременном использовании многочисленных активных датчиков может возникнуть интерференция. В целом датчики (активные и пассивные) можно разбить на три типа, в зависимости от того, регистрируют ли они расстояния до объектов, формируют изображения среды или контролируют характеристики самого робота.

В большинстве мобильных роботов используются **дальномеры**, которые представляют собой датчики, измеряющие расстояние до ближайших объектов. Одним из широко применяемых типов таких датчиков является **звуковой локатор**, известный также как ультразвуковой измерительный преобразователь. Звуковые локаторы излучают направленные звуковые волны, которые отражаются от объектов, и часть этого звука снова поступает в датчик. При этом время поступления и интенсивность такого возвратного сигнала несут информацию о расстоянии до ближайших объектов. Для автономных подводных аппаратов преимущественно используется технология подводных гидролокаторов, а на земле звуковые локаторы в основном используются для предотвращения столкновений лишь в ближайших окрестностях, поскольку эти датчики характеризуются ограниченным угловым разрешением. К числу других устройств, альтернативных по отношению к звуковым локаторам, относятся радары (в основном применяемые на воздушных судах) и лазеры. Лазерный дальномер показан на рис. 25.2.

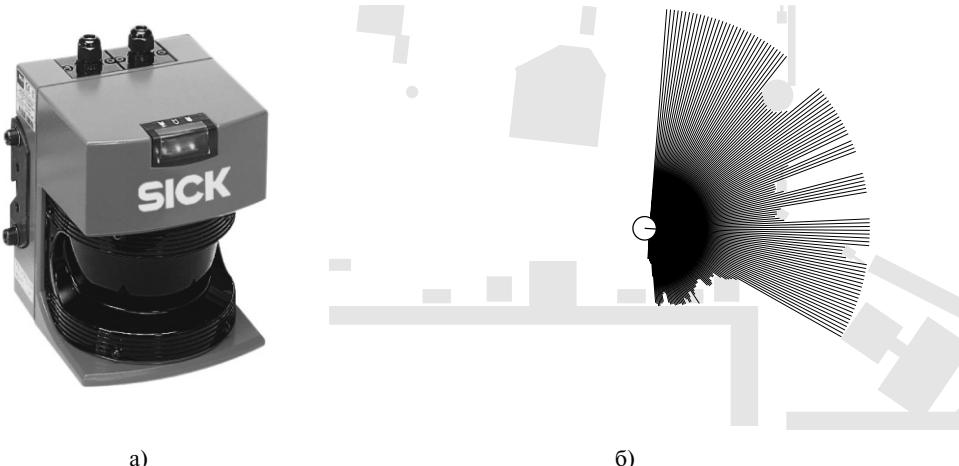


Рис. 25.2. Типичный пример датчика и его практического применения: лазерный дальномер (датчик расстояния) SICK LMS — широко применяемый датчик для мобильных роботов (а); результаты измерения расстояний, полученные с помощью горизонтально установленного датчика расстояния, спроектированные на двухмерную карту среды (б)

Некоторые датчики расстояния предназначены для измерения очень коротких или очень длинных расстояний. В число датчиков измерения коротких расстояний входят **тактильные датчики**, такие как контактные усики, контактные панели и сенсорные покрытия. На другом конце спектра находится **глобальная система позиционирования** (Global Positioning System — GPS), которая измеряет расстояние до спутников, излучающих импульсные сигналы. В настоящее время на орбите находятся свыше двух десятков спутников, каждый из которых передает сигналы на двух разных частотах. Приемники GPS определяют расстояние до этих спутников, анализируя значения фазовых сдвигов. Затем, выполняя триангуляцию сигналов от нескольких спутников, приемники GPS определяют свои абсолютные координаты на Земле с точностью до нескольких метров. В **дифференциальных системах GPS** применяется второй наземный приемник с известными координатами, благодаря чему при идеальных условиях обеспечивается точность измерения коор-

динат до миллиметра. К сожалению, системы GPS не работают внутри помещения или под водой.

Вторым важным классом датчиков являются **датчики изображения** — видеокамеры, позволяющие получать изображения окружающей среды, а также моделировать и определять характеристики среды с использованием методов машинного зрения, описанных в главе 24. В робототехнике особо важное значение имеет стереоскопическое зрение, поскольку оно позволяет получать информацию о глубине; тем не менее будущее этого направления находится под угрозой, поскольку успешно осуществляется разработка новых активных технологий получения пространственных изображений.

К третьему важному классу относятся **проприоцептивные датчики**, которые информируют робота о его собственном состоянии. Для измерения точной конфигурации робототехнического шарнира приводящие его в действие электродвигатели часто оснащаются **дешифраторами угла поворота вала**, которые позволяют определять даже небольшие приращения угла поворота вала электродвигателя. В манипуляторах роботов дешифраторы угла поворота вала способны предоставить точную информацию за любой период времени. В мобильных роботах дешифраторы угла поворота вала, которые передают данные о количестве оборотов колеса, могут использоваться для **одометрии** — измерения пройденного расстояния. К сожалению, колеса часто сдвигаются и проскальзывают, поэтому результаты одометрии являются точными только для очень коротких расстояний. Еще одной причиной ошибок при определении позиции являются внешние силы, такие как течения, воздействующие на автономные подводные аппараты, и ветры, сбивающие с курса автоматические воздушные транспортные средства. Улучшить эту ситуацию можно с использованием **инерционных датчиков**, таких как гироскопы, но даже они, применяемые без других дополнительных средств, не позволяют исключить неизбежное накопление погрешности определения положения робота.

Другие важные аспекты состояния робота контролируются с помощью **датчиков усилия** и **датчиков врачающего момента**. Без этих датчиков нельзя обойтись, если роботы предназначены для работы с хрупкими объектами или объектами, точная форма и местонахождение которых неизвестны. Представьте себе, что робототехнический манипулятор с максимальным усилием сжатия в одну тонну закручивает в патрон электрическую лампочку. При этом очень трудно предотвратить такую ситуацию, что робот приложит слишком большое усилие и раздавит лампочку. Но датчики усилия позволяют роботу ощутить, насколько крепко он держит лампочку, а датчики врачающего момента — определить, с каким усилием он ее поворачивает. Хорошие датчики позволяют измерять усилия в трех направлениях переноса и трех направлениях вращения.

## Исполнительные механизмы

Исполнительные механизмы являются теми средствами, с помощью которых роботы передвигаются и изменяют форму своего тела. Для того чтобы понять основные особенности конструкции исполнительных механизмов, необходимо вначале рассмотреть абстрактные понятия движения и формы, используя концепцию **степени свободы**. Как степень свободы мы будем рассматривать каждое независимое направление, в котором могут передвигаться либо робот, либо один из его

исполнительных механизмов. Например, твердотельный свободно движущийся робот, такой как автономный подводный аппарат, имеет шесть степеней свободы; три из них,  $(x, y, z)$ , определяют положение робота в пространстве, а три других — его угловую ориентацию по трем осям вращения, известную как качание (yaw), поворот (roll) и наклон (pitch). Эти шесть степеней свободы определяют **кинематическое состояние**<sup>2</sup> или **позу** робота. **Динамическое состояние** робота включает по одному дополнительному измерению для скорости изменения каждого кинематического измерения.

Роботы, не являющиеся твердотельными, имеют дополнительные степени свободы внутри самих себя. Например, в руке человека локоть имеет одну степень свободы (может сгибаться в одном направлении), а кисть имеет три степени свободы (может двигаться вверх и вниз, из стороны в сторону, а также вращаться). Каждый из шарниров робота также имеет 1, 2 или 3 степени свободы. Для перемещения любого объекта, такого как рука, в конкретную точку с конкретной ориентацией необходимо иметь шесть степеней свободы. Рука, показанная на рис. 25.3, *a*, имеет точно шесть степеней свободы, создаваемых с помощью пяти **поворотных шарниров**, которые формируют вращательное движение, и одного **призматического сочленения**, который формирует скользящее движение. Чтобы убедиться в том, что рука человека в целом имеет больше шести степеней свободы, можно провести простой эксперимент: положите кисть на стол и убедитесь в том, что вы еще имеете возможность поворачивать руку в локте, не меняя положения кисти на столе. Манипуляторами, имеющими больше степеней свободы, чем требуется для перевода конечного исполнительного механизма в целевое положение, проще управлять по сравнению с роботами, имеющими лишь минимальное количество степеней свободы.

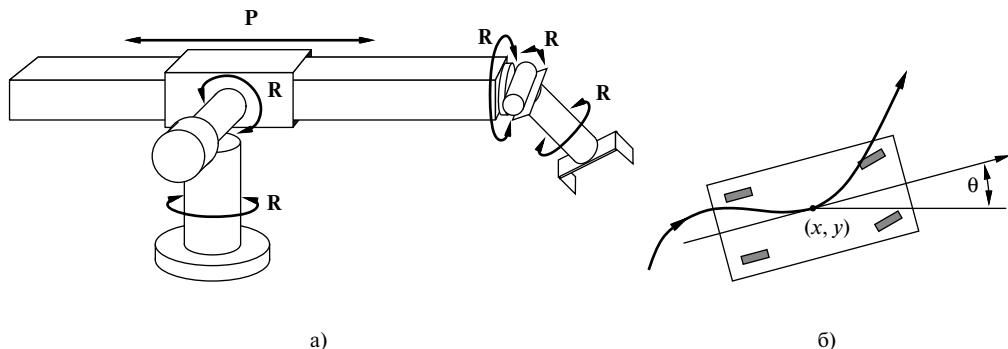


Рис. 25.3. Особенности конструкции манипулятора робота: станфордский манипулятор (*Stanford Manipulator*) — один из первых манипуляторов робота, в котором используются пять поворотных шарниров (R) и одно призматическое сочленение (P), что позволяет получить в целом шесть степеней свободы (а); траектория движения неголономного четырехколесного транспортного средства с рулевым управлением от передних колес (б)

В мобильных роботах количество степеней свободы не обязательно совпадает с количеством приводимых в действие элементов. Рассмотрим, например, обычный автомобиль: он может передвигаться вперед или назад, а также поворачиваться, что

<sup>2</sup> Термин “кинематика”, как и слово “кинематограф”, происходит от греческого корня, обозначающего движение.

соответствует двум степеням свободы. В отличие от этого кинематическая конфигурация автомобиля является трехмерной — на открытой плоской поверхности можно легко перевести автомобиль в любую точку ( $x, y$ ), с любой ориентацией (см. рис. 25.3, б). Таким образом, автомобиль имеет три **эффективные степени свободы**, но две **управляемые степени свободы**. Робот называется **неголономным**, если он имеет больше эффективных степеней свободы, чем управляемых степеней свободы, и **голономным**, если эти два значения совпадают. Голономные роботы проще в управлении (было бы намного легче припарковать автомобиль, способный двигаться не только вперед и назад, но и в стороны), однако голономные роботы являются также механически более сложными. Большинство манипуляторов роботов являются голономными, а большинство мобильных роботов — неголономными.

В мобильных роботах применяется целый ряд механизмов для перемещения в пространстве, включая колеса, гусеницы и ноги. Роботы с **дифференциальным приводом** оборудованы расположеннымми с двух сторон независимо активизируемыми колесами (или гусеницами, как в армейском танке). Если колеса, находящиеся с обеих сторон, врашаются с одинаковой скоростью, то робот движется по прямой. Если же они врашаются в противоположных направлениях, то робот поворачивается на месте. Альтернативный вариант состоит в использовании **синхронного привода**, в котором каждое колесо может вращаться и поворачиваться вокруг вертикальной оси. Применение такой системы привода вполне могло бы привести к хаотическому перемещению, если бы не использовалось такое ограничение, что все пары колес поворачиваются в одном направлении и врачаются с одинаковой скоростью. И дифференциальный, и синхронный приводы являются неголономными. В некоторых более дорогостоящих роботах используются голономные приводы, которые обычно состоят из трех или большего количества колес, способных поворачиваться и вращаться независимо друг от друга.

Ноги, в отличие от колес, могут использоваться для передвижения не по плоской поверхности, а по местности, характеризующейся очень грубым рельефом. Тем не менее на плоских поверхностях ноги как средства передвижения значительно уступают колесам, к тому же задача создания для них механической конструкции является очень сложной. Исследователи в области робототехники предприняли попытки разработать конструкции с самым разным количеством ног, начиная от одной ноги и заканчивая буквально десятками. Были разработаны роботы, оборудованные ногами для ходьбы, бега и даже прыжков (как показано на примере шагающего робота на рис. 25.4, а). Этот робот является **динамически устойчивым**; это означает, что он может оставаться в вертикальном положении, только непрерывно двигаясь. Робот, способный оставаться в вертикальном положении, не двигая ногами, называется **статически устойчивым**. Робот является статически устойчивым, если центр его тяжести находится над многоугольником, охваченным его ногами.

В мобильных роботах других типов для передвижения используются иные, чрезвычайно разнообразные механизмы. В летательных аппаратах обычно применяются пропеллеры или турбины. Роботизированные дирижабли держатся в воздухе за счет тепловых эффектов. В автономных подводных транспортных средствах часто используются подруливающие устройства, подобные тем, которые устанавливаются на подводных лодках.

Для того чтобы робот мог функционировать, ему недостаточно быть оборудованным только датчиками и исполнительными механизмами. Полноценный робот дол-

жен также иметь источник энергии для привода своих исполнительных механизмов. Для приведения в действие манипулятора и для передвижения чаще всего используются **электродвигатели**; определенную область применения имеют также **пневматические приводы**, в которых используется сжатый газ, и **гидравлические приводы**, в которых используется жидкость под высоким давлением. Кроме того, в большинстве роботов имеются некоторые средства цифровой связи наподобие беспроводной сети. Наконец, робот должен иметь жесткий корпус, на который можно было бы навесить все эти устройства, а также, фигурально выражаясь, держать при себе паяльник, на тот случай, что его оборудование перестанет работать.

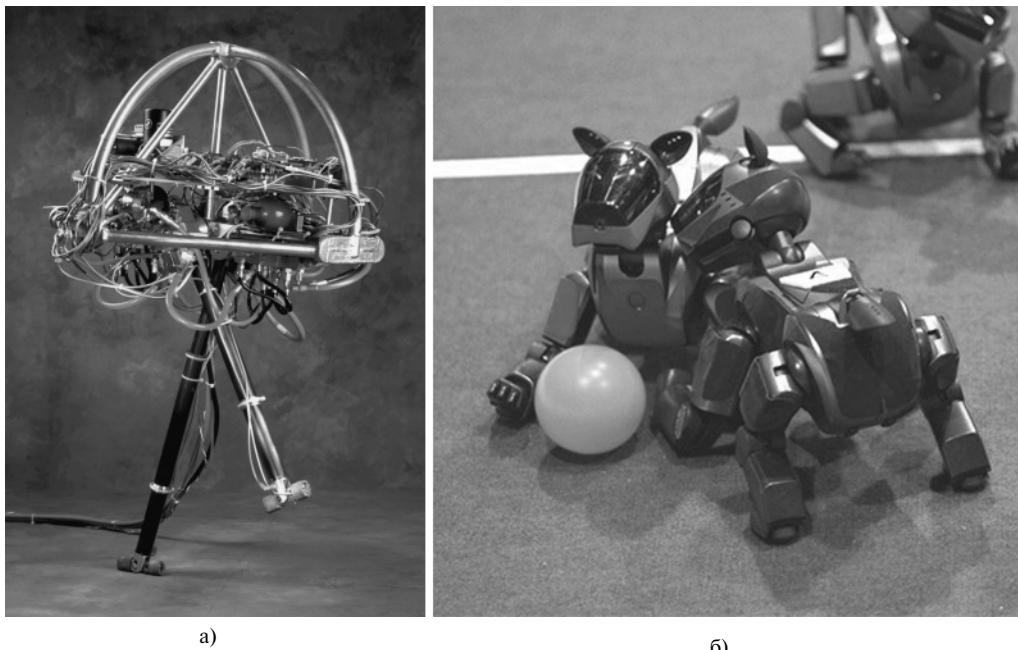


Рис. 25.4. Примеры роботов, передвигающихся с помощью ног: один из шагающих роботов Марка Рэйберта (Marc Raibert) в движении (а); роботы AIBO компании Sony, играющие в футбол (© от 2001 года, федерация RoboCup) (б)

### 25.3. ВОСПРИЯТИЕ, ОСУЩЕСТВЛЯЕМОЕ РОБОТАМИ

Робототехническое восприятие — это процесс, в ходе которого роботы отображают результаты сенсорных измерений на внутренние структуры представления среды. Задача восприятия является сложной, поскольку информация, поступающая от датчиков, как правило, зашумлена, а среда является частично наблюдаемой, не-предсказуемой и часто динамической. В качестве эмпирического правила можно руководствоваться тем, что качественные внутренние структуры представления обладают тремя свойствами: содержат достаточно информации для того, чтобы робот мог принимать правильные решения, построены так, чтобы их можно было эффективно обновлять, и являются естественными в том смысле, что внутренние переменные соответствуют естественным состояниям в физическом мире.

В главе 15 было показано, что модели перехода и восприятия для частично наблюдаемой среды могут быть представлены с помощью фильтров Калмана, скрытых марковских моделей и динамических байесовских сетей; кроме того, в указанной главе были описаны и точные, и приближенные алгоритмы обновления **доверительного состояния** — распределения апостериорных вероятностей по переменным состояния среды. К тому же в главе 15 было приведено несколько динамических моделей байесовских сетей для этого процесса. А при решении робототехнических задач в модель в качестве наблюдаемых переменных обычно включают собственные прошлые действия робота (пример такой сети см. на рис. 17.7). На рис. 25.5 показана система обозначений, используемая в данной главе:  $\mathbf{x}_t$  — это состояние среды (включая робот) во время  $t$ ;  $\mathbf{z}_t$  — результаты наблюдений, полученные во время  $t$ ;  $A_t$  — действие, предпринятое после получения этих результатов наблюдения.

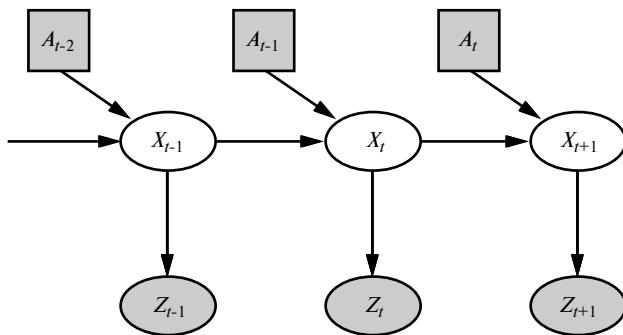


Рис. 25.5. Процесс робототехнического восприятия, рассматриваемый как временной алгоритмический вывод на основании последовательностей действий и измерений, который демонстрируется на примере динамической байесовской сети

Задача **фильтрации**, или обновления доверительного состояния, по сути является такой же, как и в главе 15. Эта задача состоит в том, что должно быть вычислено новое доверительное состояние  $P(\mathbf{x}_{t+1} | \mathbf{z}_{1:t+1}, a_{1:t})$  на основании текущего доверительного состояния  $P(\mathbf{x}_t | \mathbf{z}_{1:t}, a_{1:t-1})$  и нового наблюдения  $\mathbf{z}_{t+1}$ . Принципиальные различия по сравнению с указанной главой состоят в следующем: во-первых, результаты вычислений явно обусловлены не только действиями, но и наблюдениями, и, во-вторых, теперь приходится иметь дело с непрерывными, а не с дискретными переменными. Таким образом, необходимо следующим образом откорректировать рекурсивное уравнение фильтрации (15.3) для использования в нем интеграции, а не суммирования:

$$\begin{aligned} P(\mathbf{x}_{t+1} | \mathbf{z}_{1:t+1}, a_{1:t}) \\ = \alpha P(\mathbf{z}_{t+1} | \mathbf{x}_{t+1}) \int P(\mathbf{x}_{t+1} | \mathbf{x}_t, a_t) P(\mathbf{x}_t | \mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t \quad (25.1) \end{aligned}$$

Это уравнение показывает, что апостериорное распределение вероятностей по переменным состояния  $\mathbf{x}$  во время  $t+1$  вычисляется рекурсивно на основании соответствующей оценки, полученной на один временной шаг раньше.

В этих вычислениях участвуют данные о предыдущем действии  $a_t$  и о текущих сенсорных измерениях  $\mathbf{z}_{t+1}$ . Например, если цель заключается в разработке робота, играющего в футбол, то  $\mathbf{x}_{t+1}$  может представлять местонахождение футбольного мяча относительно робота. Распределение апостериорных вероятностей  $P(\mathbf{x}_t | \mathbf{z}_{1:t}, a_{1:t-1})$  — это распределение вероятностей по всем состояниям, отражающее все, что известно о прошлых результатах сенсорных измерений и об управляющих воздействиях. Уравнение 25.1 показывает, как рекурсивно оценить это местонахождение, инкрементно развертывая вычисления и включая в этот процесс данные сенсорных измерений (например, изображения с видеокамеры) и команды управления движением робота. Вероятность  $P(\mathbf{x}_{t+1} | \mathbf{x}_t, a_t)$  называется **моделью перехода**, или **моделью движения**, а вероятность  $P(\mathbf{z}_{t+1} | \mathbf{x}_{t+1})$  представляет собой **модель восприятия**.

## Локализация

**Локализация** — это универсальный пример робототехнического восприятия. Она представляет собой задачу определения того, где что находится. Локализация — одна из наиболее распространенных задач восприятия в робототехнике, поскольку знания о местонахождении объектов и самого действующего субъекта являются основой любого успешного физического взаимодействия. Например, роботы, относящиеся к типу манипуляторов, должны иметь информацию о местонахождении объектов, которыми они манипулируют. А роботы, передвигающиеся в пространстве, должны определять, где находятся они сами, чтобы прокладывать путь к целевым местонахождениям.

Существуют три разновидности задачи локализации с возрастающей сложностью. Если первоначальная поза локализуемого объекта известна, то локализация сводится к задаче **отслеживания траектории**. Задачи отслеживания траектории характеризуются ограниченной неопределенностью. Более сложной является задача **глобальной локализации**, в которой первоначальное местонахождение объекта полностью неизвестно. Задачи глобальной локализации преобразуются в задачи отслеживания траектории сразу после локализации искомого объекта, но в процессе их решения возникают также такие этапы, когда роботу приходится учитывать очень широкий перечень неопределенных состояний. Наконец, обстоятельства могут сыграть с роботом злую шутку и произойдет “похищение” (т.е. внезапное исчезновение) объекта, который он пытался локализовать. Задача локализации в таких неопределенных обстоятельствах называется **задачей похищения**. Ситуация похищения часто используется для проверки надежности метода локализации в крайне неблагоприятных условиях.

В целях упрощения предположим, что робот медленно движется на плоскости и что ему дана точная карта среды (пример подобной карты показан на рис. 25.7). Поза такого мобильного робота определяется двумя декартовыми координатами со значениями  $x$  и  $y$ , а также его угловым направлением со значением  $\theta$ , как показано на рис. 25.6, *a*. (Обратите внимание на то, что исключены соответствующие скорости, поэтому рассматриваемая модель скорее является кинематической, а не динамической.) Если эти три значения будут упорядочены в виде вектора, то любое конкретное состояние определится с помощью соотношения  $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$ .

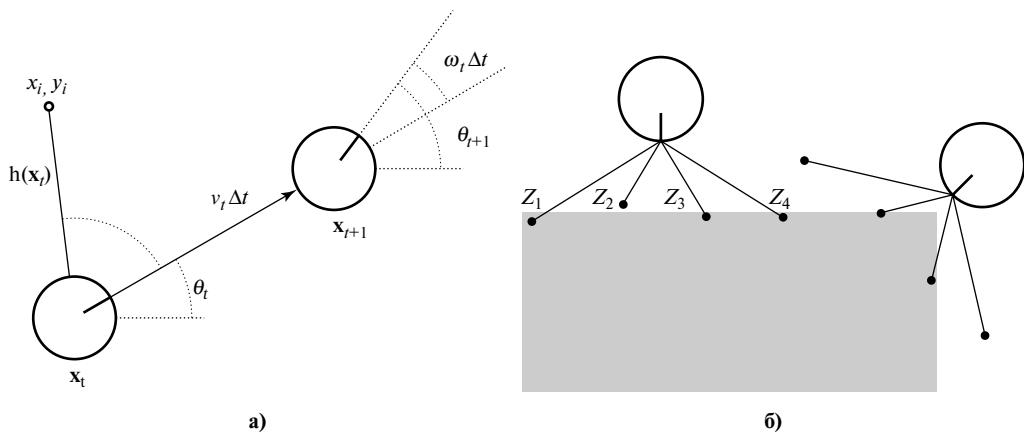


Рис. 25.6. Пример применения карты среды: упрощенная кинематическая модель мобильного робота. Робот показан в виде кружка с отметкой, обозначающей переднее направление. Показаны значения позиции и ориентации в моменты времени  $t$  и  $t+1$ , а обновления обозначены соответственно термами  $v_t \Delta t$  и  $\omega_t \Delta t$ . Кроме того, приведена отметка с координатой  $(x_i, y_i)$ , наблюдаемая во время  $t$  (а); модель датчика расстояния. Показаны две позы робота, соответствующие заданным результатам измерения расстояний  $(z_1, z_2, z_3, z_4)$ . Гораздо более вероятным является предположение, что эти результаты измерения расстояний получены в позе, показанной слева, а не справа (б)

В этой кинематической аппроксимации каждое действие состоит из “мгновенной” спецификации двух скоростей — скорости переноса  $v_t$  и скорости вращения  $\omega_t$ . Для небольших временных интервалов  $\Delta t$  грубая детерминированная модель движения таких роботов задается следующим образом:

$$\hat{\mathbf{x}}_{t+1} = f(\mathbf{x}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{x}_t + \begin{pmatrix} v_t \Delta t \cos \theta_t \\ v_t \Delta t \sin \theta_t \\ \omega_t \Delta t \end{pmatrix}$$

Обозначение  $\hat{\mathbf{x}}$  относится к детерминированному предсказанию состояния. Безусловно, поведение физических роботов является довольно непредсказуемым. Такая ситуация обычно моделируется гауссовым распределением со средним  $f(\mathbf{x}_t, v_t, \omega_t)$  и ковариацией  $\Sigma_x$  (математическое определение приведено в приложении А).

$$\mathbf{P}(\mathbf{x}_{t+1} | \mathbf{x}_t, v_t, \omega_t) = N(\hat{\mathbf{x}}_{t+1}, \Sigma_x)$$

Затем необходимо разработать модель восприятия. Рассмотрим модели восприятия двух типов. В первой из них предполагается, что датчики обнаруживают стабильные, различимые характеристики среды, называемые **отметками**. Для каждой отметки они сообщают дальность и азимут. Предположим, что состояние робота определяется выражением  $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$  и он принимает информацию об отметке, местонахождение которой, как известно, определяется координатами  $(x_i, y_i)^\top$ . При отсутствии шума дальность и азимут можно вычислить с помощью простого геометрического соотношения (см. рис. 25.6, а). Точное предсказание наблюдаемых значений дальности и азимута может быть выполнено с помощью следующей формулы:

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_t - y_i}{x_t - x_i} - \theta_t \end{pmatrix}$$

Еще раз отметим, что полученные результаты измерений искажены шумом. Для упрощения можно предположить наличие гауссова шума с ковариацией  $\Sigma_z$ :

$$P(\mathbf{z}_t | \mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \Sigma_z)$$

Для дальномеров такого типа, как показаны на рис. 25.2, часто более приемлемой является немного другая модель восприятия. Такие датчики вырабатывают вектор значений дальности  $\mathbf{z}_t = (z_1, \dots, z_M)^\top$ , в каждом из которых азимуты являются фиксированными по отношению к роботу. При условии, что дана поза  $\mathbf{x}_t$ , допустим, что  $\hat{z}_j$  — точное расстояние вдоль направления  $j$ -го луча от  $\mathbf{x}_t$  до ближайшего препятствия. Как и в описанном выше случае, эти результаты могут быть искажены гауссовым шумом. Как правило, предполагается, что погрешности для различных направлений лучей независимы и заданы в виде идентичных распределений, поэтому имеет место следующая формула:

$$P(\mathbf{z}_t | \mathbf{x}_t) = \alpha \prod_{j=1}^M e^{-(z_j - \hat{z}_j)^2 / 2\sigma^2}$$

На рис. 25.6, б показан пример четырехлучевого дальномера и двух возможных поз робота, одну из которых на полном основании можно рассматривать как позу, в которой были получены рассматриваемые результаты измерения дальностей, а другую — нет. Сравнивая модель измерения дальностей с моделью отметок, можно убедиться в том, что модель измерения дальностей обладает преимуществом в том, что не требует идентификации отметки для получения возможности интерпретировать результаты измерения дальностей; и действительно, как показано на рис. 25.6, б, робот направлен в сторону стены, не имеющей характерных особенностей. С другой стороны, если бы перед ним была видимая, четко идентифицируемая отметка, то робот мог бы обеспечить немедленную локализацию.

В главе 15 описаны фильтр Калмана, позволяющий представить доверительное состояние в виде одного многомерного гауссова распределения, и фильтр частиц, который представляет доверительное состояние в виде коллекций частиц, соответствующих состоянию. В большинстве современных алгоритмов локализации используется одно из этих двух представлений доверительного состояния робота,  $P(\mathbf{x}_t | \mathbf{z}_{1:t}, a_{1:t-1})$ .

Локализация с использованием фильтрации частиц называется **локализацией Монте-Карло**, или сокращенно MCL (Monte Carlo Localization). Алгоритм MCL идентичен алгоритму фильтрации частиц, приведенному в листинге 15.3; достаточно лишь предоставить подходящую модель движения и модель восприятия. Одна из версий алгоритма, в которой используется модель измерения дальностей, приведена в листинге 25.1. Работа этого алгоритма продемонстрирована на рис. 25.7, где показано, как робот определяет свое местонахождение в офисном здании. На первом изображении частицы распределены равномерно согласно распределению априорных вероятностей, показывающему наличие глобальной неопределенности в отношении положения робота. На втором изображении показано, как поступает первый

ряд результатов измерений и частицы формируют кластеры в областях с высоким распределением апостериорных доверительных состояний. А на третьем изображении показано, что поступило достаточное количество результатов измерений, чтобы переместить все частицы в одно место.

**Листинг 25.1. Алгоритм локализации Монте-Карло, в котором используется модель восприятия результатов измерения дальностей с учетом наличия независимого шума**

---

```

function Monte-Carlo-Localization( $a$ ,  $z$ ,  $N$ ,  $model$ ,  $map$ ) returns множество
    выборок  $S$ 
    inputs:  $a$ , предыдущая команда приведения робота в движение
     $z$ , результаты измерения дальностей с  $M$  отсчетами  $z_1, \dots, z_M$ 
     $N$ , количество сопровождаемых выборок
     $model$ , вероятностная модель среды с данными о предыдущей
        позе  $\mathbf{P}(\mathbf{x}_0)$ , моделью движения  $\mathbf{P}(\mathbf{x}_1|\mathbf{x}_0, A_0)$  и моделью
            шума для датчика расстояний  $P(Z|\hat{Z})$ 
     $map$ , двухмерная карта среды
    static:  $S$ , вектор выборок с размером  $N$ , первоначально вырабатываемый
        из  $\mathbf{P}(\mathbf{x}_0)$ 
    local variables:  $W$ , вектор весов с размером  $N$ 

    for  $i = 1$  to  $N$  do
         $S[i] \leftarrow$  выборка из  $\mathbf{P}(\mathbf{x}_1|\mathbf{x}_0=S[i], A_0=a)$ 
         $W[i] \leftarrow 1$ 
        for  $j = 1$  to  $M$  do
             $\hat{z} \leftarrow$  Exact-Range( $j, S[i], map$ )
             $W[i] \leftarrow W[i] \cdot P(Z=z_j | \hat{Z}=\hat{z})$ 
     $S \leftarrow$  Weighted-Sample-With-Replacement( $N, S, W$ )
    return  $S$ 

```

---

Еще один важный способ локализации основан на применении фильтра Калмана. Фильтр Калмана представляет апостериорную вероятность  $\mathbf{P}(\mathbf{x}_t | \mathbf{z}_{1:t}, a_{1:t-1})$  с помощью гауссова распределения. Среднее этого гауссова распределения будет обозначено  $\mu_t$ , а его ковариация —  $\Sigma_t$ . Основным недостатком использования гауссовых доверительных состояний является то, что они замкнуты только при использовании линейных моделей движения  $f$  и линейных моделей измерения  $h$ . В случае нелинейных  $f$  или  $h$  результат обновления фильтра обычно не является гауссовым. Таким образом, алгоритмы локализации, в которых используется фильтр Калмана, ~~линеаризуют~~ модели движения и восприятия. *Линеаризацией* называется локальная аппроксимация нелинейной функции с помощью линейной.

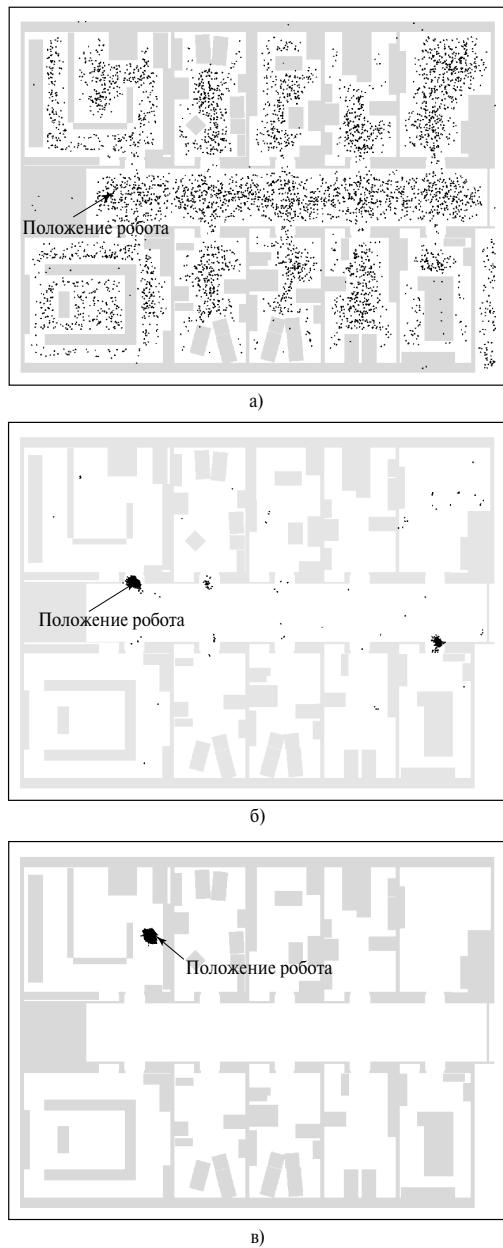


Рис. 25.7. Метод локализации Монте-Карло, основанный на применении алгоритма фильтрации частиц для локализации мобильного робота: первоначальное состояние, глобальная неопределенность (а); приблизительно бимодальное состояние неопределенности после прохождения по (симметричному) коридору (б); унимодальное состояние неопределенности после перехода в офис, отличимый от других (в)

На рис. 25.8 показано применение понятия линеаризации для (одномерной) модели движения робота. В левой части показана нелинейная модель движения  $f(\mathbf{x}_t, a_t)$  (параметр  $a_t$  на этом графике не показан, поскольку он не играет никакой роли в этой линеаризации). В правой части эта функция аппроксимируется линейной функцией  $f(\mathbf{x}_t, a_t)$ . График этой линейной функции проходит по касательной к кривой  $f$  в точке  $\mu_t$ , которая определяет среднюю оценку состояния во время  $t$ . Такая линеаризация называется **разложением в ряд Тейлора** (первой степени). Фильтр Калмана, линеаризующий функции  $f$  и  $h$  с помощью разложения в ряд Тейлора, называется **расширенным фильтром Калмана** (или Extended Kalman Filter — EKF). На рис. 25.9 показана последовательность оценок, полученных роботом, который действует под управлением алгоритма локализации на основе расширенного фильтра Калмана. По мере передвижения робота неопределенность в оценке его местонахождения возрастает, как показано с помощью эллипсов погрешностей. Но как только робот начинает получать данные о дальности и азимуте до отметки с известным местонахождением, его погрешность уменьшается. Наконец, погрешность снова возрастает, как только робот теряет отметку из виду. Алгоритмы EKF действуют успешно, если отметки являются легко идентифицируемыми. Еще один вариант состоит в том, что распределение апостериорных вероятностей может быть мультимодальным, как показано на рис. 25.7, б. Задача, для решения которой требуется знать идентификацию отметок, представляет собой пример задачи **ассоциации данных**, которая обсуждалась в конце главы 15.

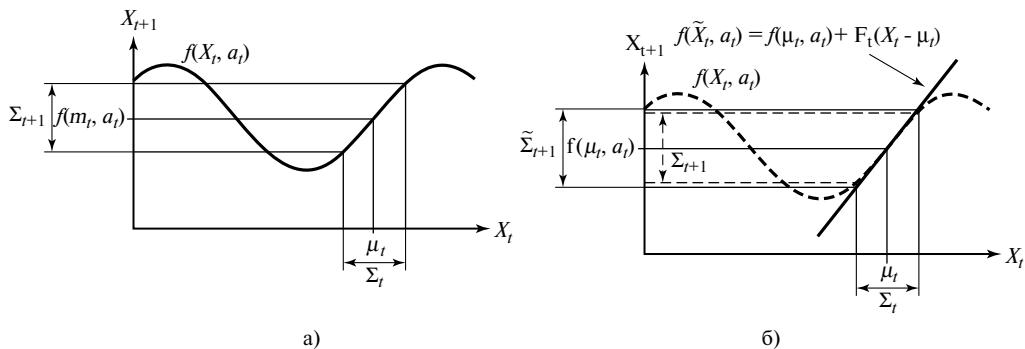


Рис. 25.8. Одномерная иллюстрация линеаризованной модели движения: функция  $f$ , проекция среднего  $\mu_t$  и интервал ковариации (основанный на  $\Sigma_t$ ) во время  $t+1$  (а); линеаризованная версия представляет собой касательную к кривой функции  $f$  при значении  $\mu_t$ . Проекция среднего  $\mu_t$  определена правильно. Однако проекция ковариации  $\Sigma_{t+1}$  отличается от  $\Sigma_{t+1}$  (б)

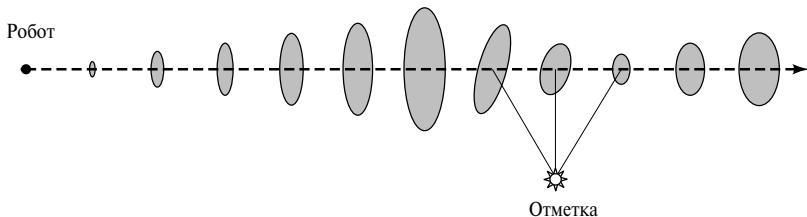


Рис. 25.9. Пример локализации с использованием расширенного фильтра Калмана. Робот движется по прямой. По мере его продвижения неопределенность постепенно возрастает, как показано с помощью эллипсов погрешностей. А после обнаружения роботом отметки с известной позицией неопределенность уменьшается

## Составление карты

До сих пор в этой главе рассматривалась задача локализации одного объекта. Но в робототехнике поиск часто осуществляется в целях локализации сразу нескольких объектов. Классическим примером такой задачи является составление карты с помощью робота. Представьте себе робота, которому не дана карта его среды. Вместо этого он вынужден составлять такую карту самостоятельно. Безусловно, человечество добилось потрясающих успехов в искусстве составления карт, описывающих даже такие крупные объекты, как вся наша планета. Поэтому одна из задач, присущих робототехнике, состоит в создании алгоритмов, позволяющих роботам решать аналогичную задачу.

В литературе задачу составления карты роботом часто называют задачей **одновременной локализации и составления карты**, сокращенно обозначая ее как SLAM (Simultaneous Localization And Mapping). Робот не только обязан составить карту, но и должен сделать это, изначально не зная, где он находится. SLAM — одна из наиболее важных задач в робототехнике. Мы рассмотрим ту версию этой задачи, в которой среда является фиксированной. Даже этот более простой вариант задачи с большим трудом поддается решению; но положение становится значительно сложнее, когда в среде допускается возникновение изменений в ходе перемещения по ней робота.

С точки зрения статистического подхода задача составления карты сводится к задаче байесовского алгоритмического вывода, так же как и локализация. Если, как и прежде, карта будет обозначаться через  $M$ , а поза робота во время  $t$  — через  $\mathbf{x}_t$ , то можно переформулировать уравнение 25.1, чтобы включить данные обо всей карте в выражение для апостериорной вероятности:

$$\begin{aligned} \mathbf{P}(\mathbf{x}_{t+1}, M | \mathbf{z}_{1:t+1}, a_{1:t}) \\ = \alpha \mathbf{P}(\mathbf{z}_{t+1} | \mathbf{x}_{t+1}, M) \int \mathbf{P}(\mathbf{x}_{t+1} | \mathbf{x}_t, a_t) \mathbf{P}(\mathbf{x}_t, M | \mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t \end{aligned}$$

На основании этого уравнения фактически можно сделать некоторые благоприятные для нас выводы: распределения условных вероятностей, необходимые для включения данных о действиях и измерениях, по существу являются такими же, как и в задаче локализации робота. Единственная предосторожность связана с тем, что новое пространство состояний (пространство всех поз робота и всех карт) имеет гораздо больше измерений. Достаточно представить себе, что принято решение изобразить конфигурацию всего здания с фотографической точностью. Для этого, по-видимому, потребуются сотни миллионов чисел. Каждое число будет представлять собой случайную переменную и вносить свой вклад в формирование чрезвычайно высокой размерности пространства состояний. Эта задача еще в большей степени усложняется в связи с тем фактом, что робот может даже не знать заранее о том, насколько велика его среда. Это означает, что ему придется динамически корректировать размерность  $M$  в процессе составления карты.

По-видимому, одним из наиболее широко применяемых методов решения задачи SLAM является EKF. Обычно этот метод используется в сочетании с моделью восприятия данных об отметках и требует, чтобы все отметки были различимыми. В предыдущем разделе апостериорная оценка была представлена с помощью гауссова распределения со средним  $\mu_t$  и ковариацией  $\Sigma_t$ . При использовании для решения

задачи SLAM подхода, основанного на методе EKF, это распределение апостериорных вероятностей снова становится гауссовым, но теперь среднее  $\mu_t$  выражается в виде вектора с гораздо большим количеством измерений. В нем представлена не только поза робота, но и местонахождение всех характеристик (или отметок) на карте. Если количество таких характеристик равно  $n$ , то вектор будет иметь размерность  $2n+3$  (два значения требуются для указания местонахождения отметки и три — для указания позы робота). Следовательно, матрица  $\Sigma_t$  имеет размерность  $(2n+3) \times (2n+3)$  и следующую структуру:

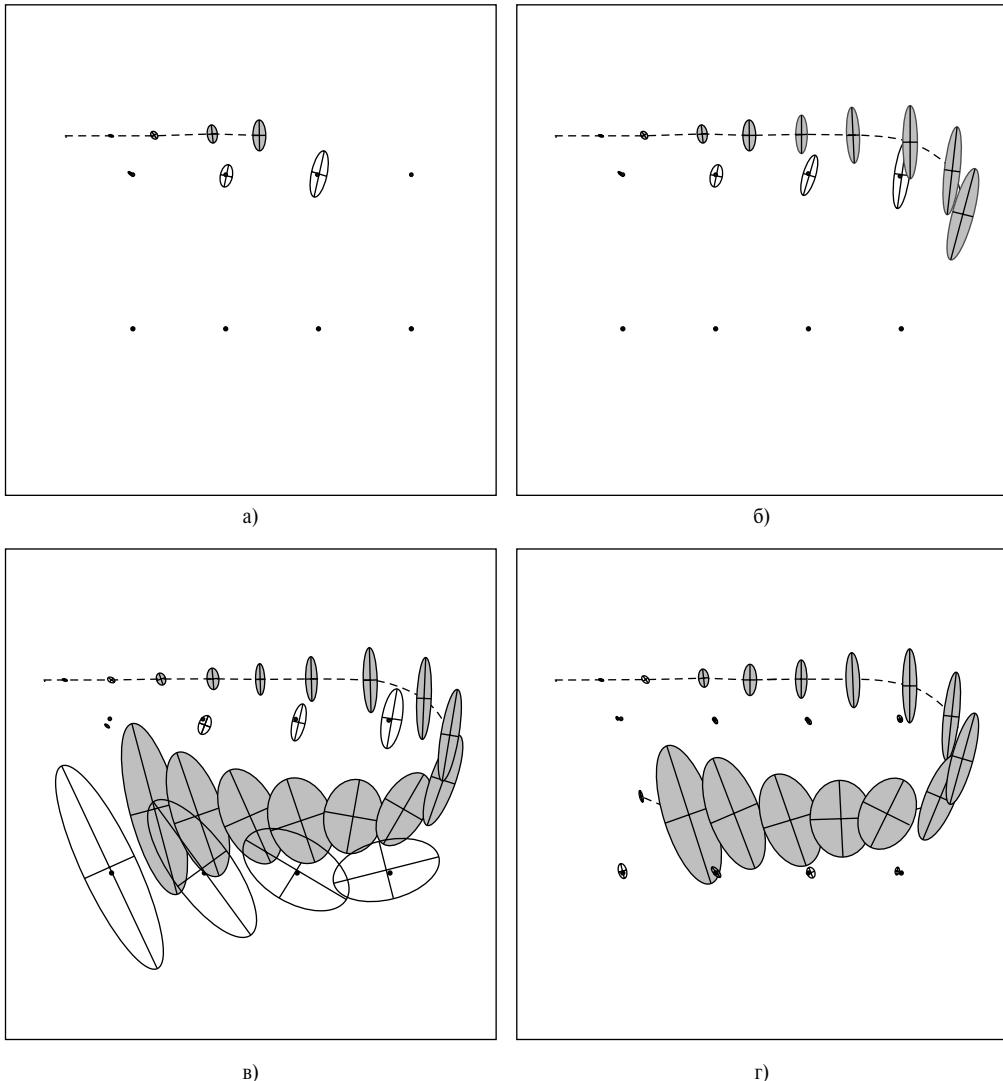
$$\Sigma_t = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{xm}^T & \Sigma_{mm} \end{pmatrix} \quad (25.2)$$

В этом уравнении  $\Sigma_{xx}$  — ковариация данных о позе робота, которая уже рассматривалась в контексте локализации;  $\Sigma_{xm}$  — матрица с размерами  $3 \times 2n$ , которая выражает корреляцию между характеристиками на карте и координатами робота. Наконец,  $\Sigma_{mm}$  — это матрица с размерами  $2n \times 2n$ , которая задает ковариацию характеристик на карте, включая все парные корреляции. Поэтому потребность в памяти для алгоритмов, основанных на методе EKF, измеряется квадратичной зависимостью от  $n$  (количество характеристик на карте), а время обновления также определяется квадратичной зависимостью от  $n$ .

Прежде чем перейти к изучению математических выкладок, рассмотрим решение задачи по методу EKF на графиках. На рис. 25.10 показано, как робот движется в среде с восемью отметками, расположенными в два ряда по четыре отметки каждый. Первоначально робот не имеет информации о том, где находятся отметки. Предполагается, что каждая отметка имеет другой цвет, и робот может надежно отличать их друг от друга. Робот начинает двигаться влево, в заранее заданном направлении, но постепенно теряет уверенность в том, есть ли у него достоверная информация о своем местонахождении. Эта ситуация показана на рис. 25.10, *a* с помощью эллипсов погрешности, ширина которых возрастает по мере дальнейшего передвижения робота. Движущийся робот получает данные о дальности и азимуте до ближайших отметок, а эти наблюдения используются для получения оценок местонахождения таких отметок. Естественно, что неопределенность в оценке местонахождения этих отметок тесно связана с неопределенностью локализации робота. На рис. 25.10, *b*, *v* показано изменение доверительного состояния робота по мере того, как он продвигается в своей среде все дальше и дальше.

Важной особенностью всех этих оценок (которую не так уж легко заметить, рассматривая приведенные графические изображения) является то, что в рассматриваемом алгоритме поддерживается единственное гауссово распределение по всем оценкам. Эллипсы погрешностей на рис. 25.10 представляют собой проекции этого гауссова распределения на подпространство координат робота и отметки. Эта многомерное гауссово распределение апостериорных вероятностей поддерживает корреляции между всеми оценками. Данное замечание приобретает важное значение при попытке понять, что происходит на рис. 25.10, *г*. На этом рисунке показано, что робот обнаруживает отметку, ранее нанесенную на карту. В результате его собственная неопределенность резко уменьшается. Такое же явление происходит и с неопределенностью всех других отметок. Указанное событие является следствием того факта, что оценка местонахождения робота и оценки местонахождений отметок имеют высокую степень корреляции в гауссовом распределении апостериорных вероятностей. Надежное выяв-

ление знаний об одной переменной (в данном случае о позе робота) автоматически приводит к уменьшению неопределенности всех других переменных.



*Рис. 25.10. Применение метода EKF для решения задачи составления карты роботом. Путь робота обозначен штриховой линией, а его оценки собственного положения — затененными эллипсами. Восемь различных отметок с неизвестными местонахождениями показаны в виде небольших точек, а оценки их местонахождения показаны в виде белых эллипсов: неопределенность робота в отношении его позиции возрастает, так же как и его неопределенность в отношении встреченных им отметок; этапы, на протяжении которых робот встречает новые отметки и наносит их на карту с возрастающей неопределенностью (а-в); робот снова встречает первую отметку, и неопределенность всех отметок уменьшается благодаря тому факту, что все эти оценки являются коррелированными (г)*

Алгоритм EKF составления карты напоминает алгоритм локализации EKF, описанный в предыдущем разделе. Основное различие между ними определяется тем, что в распределении апостериорных вероятностей учитываются дополнительные переменные отметок. Модель движения для отметок является тривиальной, поскольку они не движутся. Таким образом, для этих переменных функция  $f$  представляет собой единичную функцию, а функция измерения по существу остается такой же, как и прежде. Единственное различие состоит в том, что якобиан  $H_t$  в уравнении обновления EKF берется не только по отношению к позе робота, но также и по отношению к местонахождению отметки, которая наблюдалась во время  $t$ . Результирующие уравнения EKF являются еще более устрашающими по своей сложности, чем те, которые были сформулированы перед этим; именно по этой причине мы их здесь опускаем.

Однако существует еще одна сложность, которая до сих пор нами игнорировалась, — тот факт, что размер карты  $M$  заранее не известен. Поэтому не известно также количество элементов в окончательной оценке  $\mu_t$  и  $\Sigma_t$ . Эти данные приходится определять динамически, по мере обнаружения роботом все новых и новых отметок. Но эту проблему можно решить чрезвычайно просто — как только робот обнаруживает новую отметку, он добавляет новый элемент к распределению апостериорных вероятностей. Если же значение дисперсии этого нового элемента инициализируется очень большим числом, то результирующее распределение апостериорных вероятностей становится таким же, как если бы робот заранее знал о существовании этой отметки.

## Другие типы восприятия

Но не все средства робототехнического восприятия предназначены для локализации и составления карт. Роботов наделяют также способностями воспринимать температуру, запахи, акустические сигналы и т.д. Многие из этих измеряемых величин могут подвергаться вероятностной оценке, как и при локализации и составлении карт. Для этого требуется лишь то, чтобы такими средствами оценки служили распределения условных вероятностей, которые характеризуют эволюцию переменных состояния во времени, а также другие распределения, которые описывают связь между результатами измерений и переменными состояния.

Однако не все практически применяемые в робототехнике системы восприятия опираются на вероятностные представления. Фактически, несмотря на то, что внутреннее состояние во всех рассматриваемых выше примерах имело четкую физическую интерпретацию, такая ситуация не обязательно наблюдается в действительности. Например, представьте себе шагающего робота, который пытается перенести ногу над препятствием. Допустим, этот робот действует согласно такому правилу, что он вначале должен поднять ногу на небольшую высоту, а затем поднимать ее все выше и выше, если предыдущее значение высоты не позволяет избежать столкновения ноги с препятствием. Можно ли утверждать, что указанная в команде на выполнение этого движения высота подъема ноги является представлением некоторой физической величины в реальном мире? Возможно, что эта высота действительно как-то связана с высотой и крутизной препятствия. Но в данном случае высоту подъема ноги можно также рассматривать как вспомогательную переменную в алгоритме работы контроллера робота, лишенную непосредственного физического смысла. Подобные способы представления нередко применяются в робототехнике и вполне подходят для решения определенных задач.

В настоящее время в робототехнике ясно выражена тенденция к использованию представлений с полностью определенной семантикой. Но вероятностные методы превосходят другие подходы по своей производительности в решении многих трудных задач восприятия, таких как локализация и составление карт. Тем не менее статистические методы иногда становятся слишком громоздкими, поэтому на практике могут оказаться столь же эффективными более простые решения. Самым лучшим учителем, позволяющим понять, какого подхода действительно следует придерживаться, является опыт работы с реальными физическими роботами.

## 25.4. ПЛАНИРОВАНИЕ ДВИЖЕНИЙ

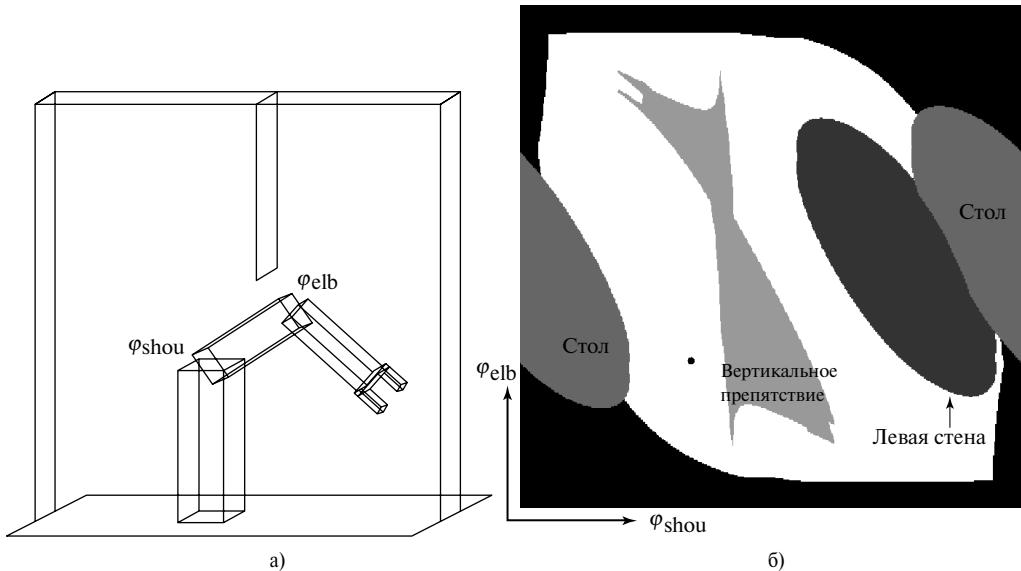
В робототехнике принятые решения в конечном итоге воплощаются в движениях исполнительных механизмов. Задача **позиционирующего движения** состоит в доставке робота или его конечного исполнительного механизма в заданную целевую позицию. Это — сложная задача, но еще сложнее задача **согласующего движения**, при выполнении которой робот движется, находясь в физическом контакте с препятствием. Примером согласующего движения является закручивание электрической лампочки манипулятором робота или подталкивание роботом ящика для его перемещения по поверхности стола.

Начнем с поиска подходящего представления, которое позволяло бы описывать и решать задачи планирования движений. Как оказалось, более удобным для работы по сравнению с исходным трехмерным пространством является **пространство конфигураций** — пространство состояний робота, определяемых положением, ориентацией и углами поворота шарниров. Задача **планирования пути** состоит в поиске пути от одной конфигурации к другой в пространстве конфигураций. В этой книге уже встречались различные версии задачи планирования пути, а в робототехнике основной характерной особенностью планирования пути является то, что в этой задаче должны рассматриваться непрерывные пространства. В литературе по робототехническому планированию пути рассматривается широкий набор различных методов, специально предназначенных для поиска путей в непрерывных пространствах с большим количеством измерений. Основные семейства применяемых при этом подходах известны под названиями **декомпозиции ячеек** и **скелетирования**. В каждом из этих подходов задача планирования непрерывного пути сводится к задаче поиска в дискретном графе на основе выявления некоторых канонических состояний и путей в свободном пространстве. Во всем данном разделе предполагается, что движения детерминированы, а информация о локализации робота является точной. В следующих разделах эти предположения будут ослаблены.

### Пространство конфигураций

Первый шаг к решению задачи управления движением робота состоит в создании подходящего представления задачи. Начнем с простого представления для простой задачи. Рассмотрим манипулятор робота, показанный на рис. 25.11, *a*. В нем имеются два шарнира, которые движутся независимо друг от друга. В результате движения шарниров изменяются координаты (*x*, *y*) локтя и захвата (манипулятор не может двигаться в направлении *z*). Это описание показывает, что конфигурацию данного

робота можно описать с помощью четырехмерных координат: использовать координаты  $(x_e, y_e)$  для обозначения местонахождения локтя относительно среды и координаты  $(x_g, y_g)$  — для обозначения местонахождения захвата. Очевидно, что эти четыре координаты полностью характеризуют состояние робота. Они составляют представление, которое принято называть представлением **рабочего пространства**, поскольку координаты робота заданы в той же системе координат, что и объекты, которыми он должен манипулировать (или столкновения с которыми должен избегать). Представления рабочего пространства хорошо подходят для проверки на предмет столкновения, особенно если робот и все объекты представлены с помощью простых многоугольных моделей.



*Рис. 25.11. Сравнение способов представления: представление рабочего пространства манипулятора робота с двумя степенями свободы; рабочее пространство представляет собой ящик с плоским препятствием в виде пластины, свисающей с потолка (а); пространство конфигураций того же робота. В этом пространстве только участки, обозначенные белым цветом, соответствуют конфигурациям, в которых не возникают столкновения с препятствиями. Точка на этом рисунке соответствует конфигурации робота, показанной слева (б)*

Но представление рабочего пространства имеет один недостаток, связанный с тем, что фактически не все координаты рабочего пространства являются достижимыми, даже в отсутствии препятствий. Это обусловлено наличием **ограничений связи** в пространстве достижимых координат рабочего пространства. Например, позиция локтя  $(x_e, y_e)$  и позиция захвата  $(x_g, y_g)$  всегда разнесены на постоянное расстояние, поскольку шарниры, соответствующие этим позициям, связаны с помощью жесткого предплечья. Планировщик движений робота с алгоритмом, определенным на координатах рабочего пространства, сталкивается с проблемой выработки таких путей, которые позволяют придерживаться указанных ограничений. Такая задача становится особенно сложной в связи с тем, что пространство состояний является непрерывным, а ограничения — нелинейными.

Как оказалось, проще осуществлять планирование на основе представления пространства конфигураций. Вместо представления состояния робота с помощью декартовых координат его элементов это состояние представляется с помощью конфигурации шарниров робота. В данном примере в конструкцию робота входят два шарнира. Поэтому его состояние может быть представлено двумя углами,  $\phi_s$  и  $\phi_e$ , относящимися соответственно к шарниру плеча и к шарниру локтя. В отсутствие каких-либо препятствий робот может свободно выбрать любое значение из пространства конфигураций. В частности, при планировании пути можно связать текущую и целевую конфигурацию прямой линией. В таком случае, следуя по этому пути, робот может просто изменять углы поворота своих шарниров с постоянной скоростью до тех пор, пока не будет достигнуто целевое местонахождение.

К сожалению, и подход на основе пространства конфигураций имеет свои недостатки. Задание для робота обычно выражается в координатах рабочего пространства, а не в координатах пространства конфигураций. Например, может потребоваться, чтобы робот поместил свой конечный исполнительный механизм в определенную координату в рабочем пространстве, возможно, с указанием также его ориентации. В связи с этим возникает вопрос: как отобразить такие координаты рабочего пространства на пространство конфигураций? Вообще говоря, проще поддается решению обратная задача — преобразование координат пространства конфигураций в координаты рабочего пространства, поскольку для этого достаточно выполнить ряд совершенно очевидных преобразований координат. Эти преобразования являются линейными для призматических шарниров и тригонометрическими для поворотных шарниров. Такую цепочку преобразований координат принято называть **кинематикой**; этот термин уже встречался при обсуждении мобильных роботов.

Обратная задача вычисления конфигурации робота, для которого задано местонахождение исполнительного механизма в координатах рабочего пространства, называется **обратной кинематикой**. Проблема вычисления обратной кинематики, как правило, является трудной, особенно для роботов со многими степенями свободы. В частности, это решение редко является уникальным. Для рассматриваемого в качестве примера манипулятора робота существуют две различные конфигурации, при которых захват занимает одни и те же координаты рабочего пространства (см. рис. 25.11).

Вообще говоря, для любого множества координат рабочего пространства этого манипулятора робота с двумя сочленениями количество обратных кинематических решений изменяется в пределах от нуля до двух. А для большинства промышленных роботов количество решений бесконечно велико. Чтобы понять, почему это возможно, достаточно представить себе, что в роботе, рассматриваемом в качестве примера, будет установлен третий, дополнительный шарнир, ось вращения которого параллельна оси существующего шарнира. В таком случае можно будет поддерживать фиксированное местонахождение (но не ориентацию!) захвата и вместе с тем свободно вращать его внутренние шарниры в большинстве конфигураций робота. Добавив еще несколько шарниров (определите, сколько именно?), можно добиться того же эффекта, поддерживая также постоянную ориентацию. Пример аналогичной ситуации уже рассматривался в данной книге, когда читателю было предложено провести “эксперимент”, положив ладонь на стол и двигая локтем. В таком случае кинематическое ограничение на позицию ладони не позволяет однозначно определить конфигурацию локтя. Иными словами, задача определения обратной кинематики

тики для сочленения “плечо–предплечье” руки с ладонью, лежащей на столе, имеет бесконечное количество решений.

Вторая проблема, возникающая при использовании представлений пространства конфигураций, связана с наличием препятствий, которые могут существовать в рабочем пространстве робота. В примере, приведенном на рис. 25.11, *a*, показано несколько таких препятствий, включая свободно свисающую с потолка полосу, которая проникает в самый центр рабочего пространства робота. В рабочем пространстве такие препятствия рассматриваются как простые геометрические формы, особенно в большинстве учебников по робототехнике, которые в основном посвящены описанию многоугольных препятствий. Но как эти препятствия выглядят в пространстве конфигураций?

На рис. 25.11, *b* показано пространство конфигураций робота, рассматриваемого в качестве примера, при той конкретной конфигурации препятствий, которая приведена на рис. 25.11, *a*. Это пространство конфигураций можно подразделить на два подпространства: пространство всех конфигураций, достижимых для робота, которое принято называть **свободным пространством**, и пространство недостижимых конфигураций, называемое **занятым пространством**. Обозначенный белым цветом участок на рис. 25.11, *b* соответствует свободному пространству. Все другие участки соответствуют занятому пространству. Различные затенения в занятом пространстве соответствуют разным объектам в рабочем пространстве робота; участки, выделенные черным цветом и окружающие все свободное пространство, соответствуют конфигурациям, в которых робот сталкивается сам с собой. Можно легко обнаружить, что подобные нарушения в работе возникают при крайних значениях углов поворота шарниров плеча или локтя. Два участка овальной формы по обе стороны от робота соответствуют столу, на котором смонтирован робот. Аналогичным образом, третий овальный участок соответствует левой стене. Наконец, наиболее интересным объектом в пространстве конфигураций является простое вертикальное препятствие, проникающее в рабочее пространство робота. Этот объект имеет любопытную форму: он чрезвычайно нелинейен, а в некоторых местах является даже вогнутым. Приложив немного воображения, читатель легко узнает форму захвата на верхнем левом конце манипулятора. Рекомендуем читателю на минуту задержаться и изучить эту важную схему. Форма рассматриваемого препятствия не так уж очевидна! Точка внутри рис. 25.11, *b* обозначает конфигурацию робота, как показано на рис. 25.11, *a*. На рис. 25.12 изображены три дополнительные конфигурации как в рабочем пространстве, так и в пространстве конфигураций. В конфигурации “conf-1” захват окружает вертикальное препятствие.

Вообще говоря, даже если рабочее пространство робота представлено с помощью плоских многоугольников, форма свободного пространства может оказаться очень сложной. Поэтому на практике обычно применяется ощупывание пространства конфигураций вместо явного его построения. Планировщик может вырабатывать конфигурацию, а затем проверять ее для определения того, находится ли она в свободном пространстве, применяя кинематику робота и определяя наличие столкновений в различных координатах рабочего пространства.

## Методы декомпозиции ячеек

В указанном выше первом подходе к планированию пути используется **декомпозиция ячеек**; иными словами, в этом методе осуществляется разложение

свободного пространства на конечное количество непрерывных участков, называемых *ячейками*. Эти участки обладают тем важным свойством, что задача планирования пути в пределах одного участка может быть решена с помощью простых средств (например, в виде передвижения по прямой линии). Таким образом, задача планирования пути преобразуется в задачу поиска в дискретном графе, во многом аналогичную задачам поиска, представленным в главе 3.

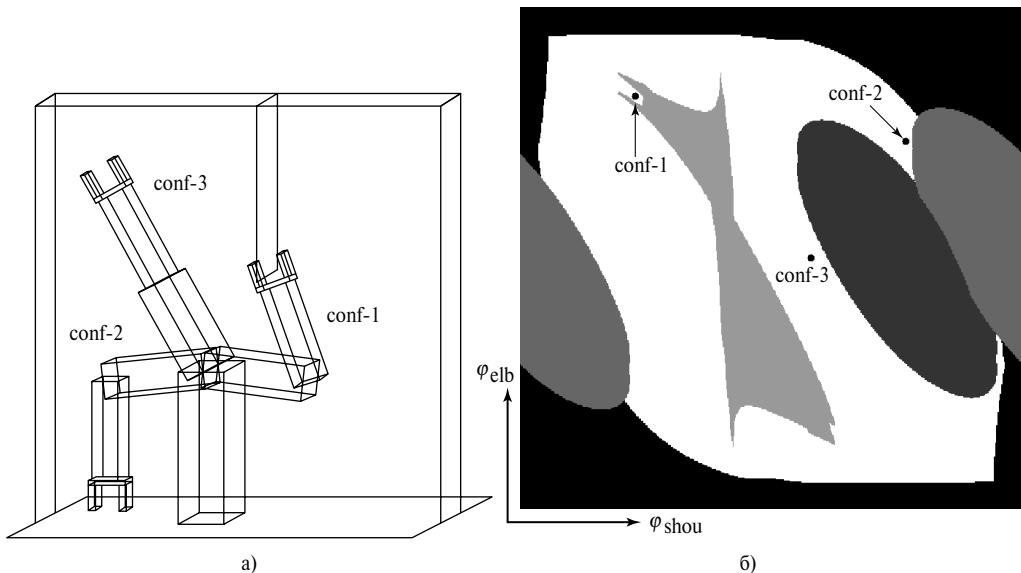
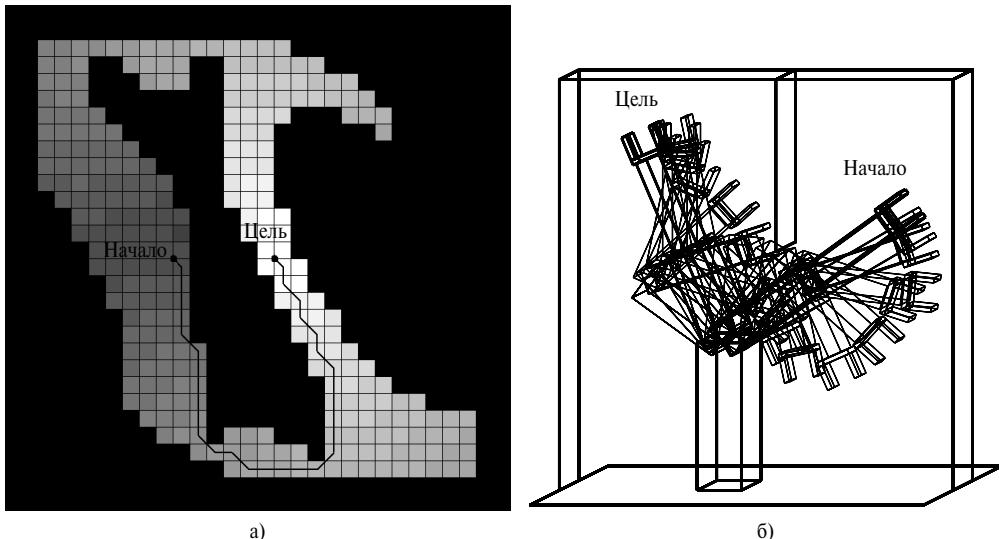


Рис. 25.12. Три конфигурации робота, показанные в рабочем пространстве и пространстве конфигураций

Простейшая декомпозиция ячеек представляет собой сетку с равномерным шагом. На рис. 25.13, а показаны декомпозиция пространства с помощью квадратной сетки и путь решения, оптимальный для сетки с этими размерами. Кроме того, на этом рисунке используется затенение в виде градаций серого цвета для обозначения стоимости каждой ячейки сетки свободного пространства, т.е. стоимости самого короткого пути от этой ячейки к цели. (Эти стоимости можно вычислить с помощью детерминированной формы алгоритма Value-Iteration, приведенного в листинге 17.1.) На рис. 25.13, б показана соответствующая траектория манипулятора в рабочем пространстве.

Такая декомпозиция имеет преимущество в том, что обеспечивает чрезвычайно простую реализацию, но характеризуется также двумя ограничениями. Во-первых, она может применяться только для пространств конфигураций с малым количеством измерений, поскольку количество ячеек сетки растет экспоненциально в зависимости от  $d$ , т.е. от количества измерений. Во-вторых, возникает проблема, обусловленная тем, что некоторые ячейки являются “смешанными”, т.е. не принадлежащими полностью ни к свободному, ни к занятому пространству. Путь, найденный в качестве решения, который включает такую ячейку, может не соответствовать действительному решению, в связи с тем что не будет существовать способа пересечения ячейки в желаемом направлении по прямой линии. В результате этого процеду-

ра планирования пути становится противоречивой. С другой стороны, если мы будем настаивать на том, чтобы использовались только полностью свободные ячейки, то процедура планирования станет неполной, в связи с тем что могут возникать случаи, в которых единственные возможные пути к цели лежат через смешанные ячейки, особенно если размер ячейки сопоставим с размерами проходов и просветов в рассматриваемом пространстве.



*Рис. 25.13. Пример применения метода декомпозиции ячеек: функция стоимости и путь, найденный с помощью аппроксимации пространства конфигураций в виде ячеек сетки (а); тот же путь, визуально представленный с помощью координат рабочего пространства (б). Обратите внимание на то, как робот сгибает свой локоть для предотвращения столкновения с вертикальным препятствием*

Существуют два способа усовершенствования метода декомпозиции ячеек, позволяющие исправить эти недостатки. Первый из них состоит в том, что допускается дальнейшее разделение смешанных ячеек, возможно, с использованием ячеек, вдвое меньших по сравнению с первоначальным размером. Такая операция может продолжаться рекурсивно до тех пор, пока не будет найден путь, полностью проходящий по свободным ячейкам. (Безусловно, этот метод может применяться, только если есть возможность определить, является ли данная конкретная ячейка смешанной, а эта операция является простой, только если границы пространства конфигураций определяются с помощью относительно простых математических описаний.) Такой метод является полным, при условии, что заданы ограничения на величину наименьшего прохода, через который должен пройти искомый путь. Хотя при этом основная часть усилий, связанных с вычислениями, сосредоточивается на наиболее сложных участках в пространстве конфигураций, данный метод все еще не позволяет добиться успешного масштабирования и расширения его на многомерные задачи, поскольку при каждом рекурсивном разбиении ячейки создаются  $2^d$  меньших ячеек. Второй способ получения полного алгоритма состоит в том, чтобы неуклонно соблюдалось требование **точной декомпозиции ячеек** свободного пространства. Этот метод должен допускать, чтобы ячейки принимали неправильную форму в тех местах, где они встречаются с границами свободного пространства, но эти формы все еще должны оставаться “простыми” в том смысле, что при их ис-

пользовании можно было легко вычислить траекторию прохождения через любую свободную ячейку. Для реализации этого метода требуется использование некоторых весьма сложных геометрических понятий, поэтому данный метод не будет рассматриваться здесь более подробно.

Рассматривая путь решения, показанный на рис. 25.13, *a*, можно заметить дополнительные сложности, которые необходимо преодолеть. Во-первых, следует отметить, что этот путь содержит произвольно острые углы; робот, движущийся с какой-либо конечной скоростью, не сможет пройти по такому пути. Во-вторых, заслуживает внимания то, что путь проходит очень близко от препятствия. Любой, кто занимается вождением автомобиля, знает, что парковочная площадка, на которой оставлено по одному миллиметру просвета с каждой стороны, в действительности вообще не годится для парковки; по той же причине следует предпочесть такие пути решения, которые не чувствительны к небольшим погрешностям движения.

Желательно максимизировать расстояние от препятствий и вместе с тем минимизировать длину пути. Этой цели можно достичь, введя понятие **поля потенциалов**. Поле потенциалов — это функция, определенная в пространстве состояний, значение которой растет пропорционально расстоянию до ближайшего препятствия. Такое поле потенциалов показано на рис. 25.14, *a*, — чем более темным цветом обозначена точка в пространстве конфигураций, тем ближе она к препятствию. При использовании в задаче планирования пути такое поле потенциалов становится дополнительным термом стоимости в уравнении оптимизации. Благодаря этому возникает интересная ситуация поиска компромисса. С одной стороны, робот стремится минимизировать длину пути к цели. С другой стороны, он пытается оставаться в стороне от препятствий, придерживаясь минимальных значений функции потенциалов. Назначив обеим целям соответствующие веса, можно найти примерно такой путь, как показано на рис. 25.14, *б*. На этом рисунке показана также функция стоимости, выведенная на основании комбинированной функции затрат, которая и в этом случае вычислена с помощью итерации по стоимостям. Очевидно, что полученный в результате путь длиннее, но вместе с тем и безопаснее.

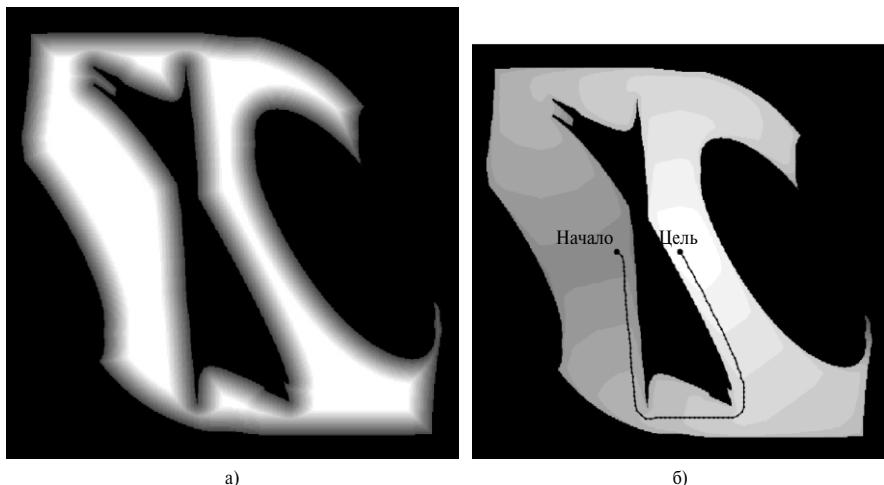
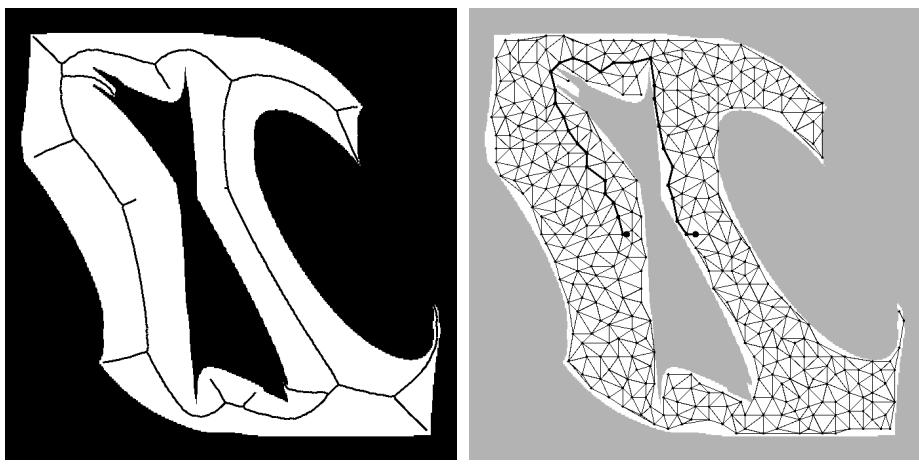


Рис. 25.14. Метод с использованием поля потенциалов: препятствующее сближению поле потенциалов отталкивает робота от препятствий (*а*); путь, найденный с помощью одновременной минимизации длины пути и потенциала (*б*)

### Методы скелетирования

Второе важное семейство алгоритмов планирования пути основано на идеи **скелетирования**. Эти алгоритмы сводят свободное пространство робота к одномерному представлению, для которого задача планирования становится проще. Такое представление с меньшим количеством измерений называется **скелетом** пространства конфигураций.

Пример применения метода скелетирования приведен на рис. 25.15: это — **линия Вороного** для свободного пространства, которая представляет собой геометрическое место всех точек, равноудаленных от двух или нескольких препятствий. Для того чтобы осуществить планирование пути с помощью линии Вороного, робот вначале переходит из текущей конфигурации в точку на линии Вороного. Можно легко показать, что такую операцию всегда можно выполнить с помощью передвижения по прямой в пространстве конфигураций. Затем робот следует по линии Вороного до тех пор, пока не достигнет точки, ближайшей к целевой конфигурации. Наконец, робот покидает линию Вороного и движется к цели. И на этом последнем этапе снова выполняется движение по прямой в пространстве конфигураций.



*Рис. 25.15. Один из примеров метода скелетирования: линия Вороного — это геометрическое место точек, равноудаленных от двух или нескольких препятствий в пространстве конфигураций (а); вероятностная дорожная карта, состоящая из 400 случайно выбранных точек в свободном пространстве (б)*

Таким образом, первоначальная задача планирования пути сводится к поиску пути на линии Вороного, которая обычно является одномерной (за исключением некоторых частных случаев) и имеет конечное количество таких точек, в которых пересекаются три или большее количество одномерных кривых. Поэтому задача поиска кратчайшего пути вдоль линии Вороного сводится к задаче поиска в дискретном графе такого же типа, как было описано в главах 3 и 4. Движение по линии Вороного может не обеспечить получение кратчайшего пути, но обнаруженные пути будут отличаться наличием максимальных расстояний от препятствий. Недостатки методов, основанных на использовании линии Вороного, состоят в том, что их сложно применять в пространствах конфигураций с большими размерностями, кроме того, при

их использовании приходится совершать слишком большие обходные маневры, если пространство конфигураций характеризуется широким размахом. К тому же может оказаться сложным вычисление линии Вороного, особенно в пространстве конфигураций, характеризующемся сложной формой препятствий.

Альтернативным по отношению к методу на основе линии Вороного является метод с использованием **вероятностной дорожной карты**. Он представляет собой такой подход к скелетированию, который позволяет определить больше возможных маршрутов и поэтому лучше подходит для пространств с широким размахом. Пример вероятностной дорожной карты показан на рис. 25.15, б. Линия, приведенная на этом рисунке, создана путем формирования случайным образом большого количества конфигураций и удаления тех из них, которые не укладываются в свободное пространство. После этого любые два узла соединяются какой-то линией, если одного из них можно “легко” достичь из другого; например, если в свободном пространстве можно перейти из одного узла в другой по прямой. Конечным итогом выполнения всех этих операций становится создание рандомизированного графа в свободном пространстве робота. Если к этому графу будут добавлены позиции начальной и целевой конфигураций робота, то задача планирования пути сводится к поиску в дискретном графе. Теоретически этот подход является неполным, поскольку при неудачном выборе случайно заданных точек может оказаться, что нельзя найти ни одного пути от начального узла до целевого. Но вероятность такой неудачи можно ограничить за счет регламентации количества формируемых точек и с учетом определенных геометрических свойств пространства конфигураций. Возможно также направить процесс выработки опорных точек в те области, где частично выполненный поиск показывает хорошие перспективы поиска приемлемого пути, действуя одновременно в двух направлениях, от начальной и от целевой позиций. После внесения всех этих усовершенствований метод планирования с помощью вероятностной дорожной карты показывает лучшую масштабируемость в условиях многомерных пространств конфигураций по сравнению с большинством других альтернативных методов планирования путей.

## 25.5. ПЛАНИРОВАНИЕ ДВИЖЕНИЙ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ

Ни в одном из алгоритмов планирования движения робота, рассмотренных выше, не шла речь о наиболее важной характерной особенности робототехнических задач — об их неопределенности. В робототехнике неопределенность возникает из-за частичной наблюдаемости среды, а также под влиянием стохастических (или не предусмотренных моделью) результатов действий робота. Кроме того, могут возникать погрешности, обусловленные использованием приближенных алгоритмов, таких как фильтрация частиц, в результате чего робот не будет получать точных данных о текущем доверительном состоянии, даже несмотря на то, что для описания стохастического характера среды применяется идеальная модель.

В большинстве современных роботов для принятия решений используются детерминированные алгоритмы, такие как различные алгоритмы планирования пути, рассматривавшиеся до сих пор. Для этой цели обычно принято извлекать данные

о **наиболее вероятном состоянии** из распределения состояний, сформированного с помощью алгоритма локализации. Преимущество этого подхода состоит лишь в том, что он способствует уменьшению объема вычислений. Трудной является даже сама задача планирования путей через пространство конфигураций, а если бы нам пришлось работать с полным распределением вероятностей по состояниям, то задача стала бы еще труднее. Поэтому игнорировать неопределенность в этих обстоятельствах можно, только если неопределенность мала.

К сожалению, игнорировать неопределенность не всегда возможно. Дело в том, что при решении некоторых задач возникает такая ситуация, что неопределенность, в условиях которой действует робот, становится слишком большой. Например, как можно использовать детерминированный планировщик пути для управления мобильным роботом, не имеющим информации о том, где он находится? Вообще говоря, если истинное состояние робота не является таковым, на которое указывает правило максимального правдоподобия, то в итоге управляющие воздействия будут далеки от оптимальных. В зависимости от величины погрешности они могут приводить ко всякого рода нежелательным эффектам, таким как столкновения с препятствиями.

В этой области робототехники нашел свое применение целый ряд методов организации работы в условиях неопределенности. Некоторые из этих методов основаны на приведенных в главе 17 алгоритмах принятия решений в условиях неопределенности. Если робот сталкивается с неопределенностью только при переходах из одного состояния в другое, но само состояние является полностью наблюдаемым, то эту задачу лучше всего можно промоделировать в виде марковского процесса принятия решения, или MDP (Markov Decision Process). Решением задачи MDP является оптимальная **политика**, с помощью которой робот может определить, что делать в каждом возможном состоянии. Таким образом, он получает возможность исправить погрешности движения всех видов, тогда как решение, полученное от детерминированного планировщика, с указанием единственного пути, может быть гораздо менее надежным. В робототехнике вместо термина политика обычно используют термин **функция навигации**. Функцию стоимости, показанную на рис. 25.13, *a* можно преобразовать в такую функцию навигации, обеспечив отслеживание градиента.

Так же как и в задачах, описанных в главе 17, задачи, рассматриваемые в настоящей главе, становятся гораздо более трудными в условиях частичной наблюдательности. Возникающая в результате задача управления роботом представляет собой **частично наблюдаемую задачу MDP**, или POMDP (partially observable MDP). В таких ситуациях робот обычно поддерживает внутреннее доверительное состояние, наподобие описанного в разделе 25.3. Решением задачи POMDP является политика, определенная на доверительных состояниях робота. Иными словами, входными данными для рассматриваемой политики является все распределение вероятностей. Это позволяет роботу основывать свое решение не только на том, что ему известно, но и на том, что неизвестно. Например, если робот действует в условиях неопределенности в отношении какой-то важной переменной состояния, он может принять рациональное в этих условиях решение и вызвать на выполнение **действие по сбору информации**. Такой подход в инфраструктуре MDP невозможен, поскольку в задачах MDP подразумевается наличие полной наблюдаемости. К сожалению, методы точного решения задач POMDP не применимы к робототехнике, поскольку не существует известных методов для непрерывных пространств. А в результате дискретиза-

ции обычно создаются такие задачи POMDP, которые слишком велики, чтобы их можно было решить с помощью известных методов. Все, что можно сделать в настоящее время, — это пытаться свести неопределенность в отношении позы к минимуму; например, в эвристике **плавания вдоль берегов** требуется, чтобы робот оставался неподалеку от известных отметок в целях уменьшения неопределенности в отношении его позы. Такая ситуация, в свою очередь, приводит к постепенному уменьшению неопределенности при нанесении на карту обнаруженных поблизости новых отметок, а это в дальнейшем позволяет роботу исследовать новые территории.

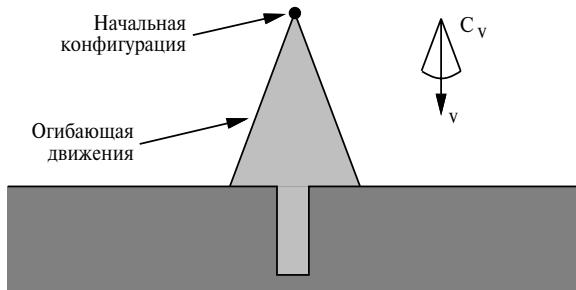
## Надежные методы

С неопределенностью можно также справиться, используя так называемые **надежные**, а не вероятностные методы. *Надежным* называется такой метод, в котором подразумевается наличие ограниченного объема неопределенности в каждом аспекте задачи, но не присваиваются вероятности значениям в пределах разрешенного интервала. *Надежным* называется такое решение, которое приводит к намеченной цели независимо от того, какие значения данных встречаются в действительности, при условии, что они находятся в пределах предполагаемого интервала. Крайней формой надежного метода является подход на основе **совместимого планирования**, описанный в главе 12, — в нем вырабатываются планы, выполнимые даже без учета информации о состоянии.

В настоящем разделе рассматривается один из надежных методов, применяемый для **планирования тонких движений** (или сокращенно FMP — Fine-Motion Planning) в задачах робототехнической сборки. Планирование тонких движений обеспечивает перемещение манипулятора робота в очень тесной близости от объекта в статической среде. Основная сложность, связанная с планированием тонких движений, состоит в том, что требуемые движения и соответствующие характеристики среды очень малы. В таких малых масштабах робот теряет возможность точно измерять или управлять своим положением, кроме того, может возникать неопределенность в отношении формы самой среды; предполагается, что все эти неопределенности ограничены. Решением задачи FMP обычно становится условный план (или политика), в котором используется обратная связь от датчиков и который гарантирует успешное выполнение во всех ситуациях, совместимых с предполагаемыми пределами неопределенности.

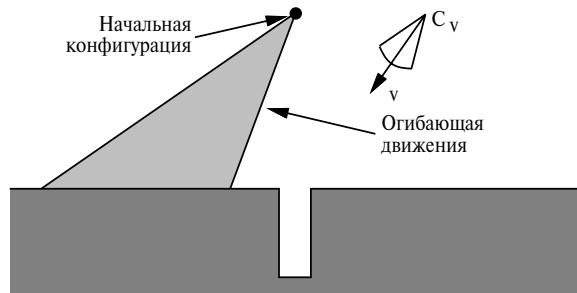
План проведения тонких движений представляет собой определение ряда **охраняемых движений**. Каждое охраняемое движение состоит, во-первых, из команды движения и, во-вторых, из условия завершения, которое представляет собой предикат, заданный на сенсорных значениях робота, и возвращает истинное значение в качестве указания на окончание охраняемого движения. Команды движения обычно задают **приспособляемые движения**, которые позволяют роботу выполнять скользящие движения, если другие команды движения вызовут столкновение с препятствием. В качестве примера на рис. 25.16 показано двухмерное пространство конфигураций с узким вертикальным отверстием. Такое пространство конфигураций может возникнуть при решении задачи вставки прямоугольного колышка в отверстие, немного превышающее его по размерам. Команды движения выполняются с постоянными скоростями. Условиями завершения являются ситуации контакта с поверхностью. Для моделирования неопределенности в процессе управления предположим, что факти-

чески движение робота происходит не в направлении, указанном в команде, а укладывается в конус  $C_v$  вокруг этого направления. На рис. 25.16 показано, что произойдет, если будет выдана команда движения с постоянной скоростью строго в вертикальном направлении из исходной области  $s$ . Из-за неопределенности в скорости робот может совершать движения в любом направлении в пределах конической огибающей; возможно, что это приведет к попаданию в отверстие, но с большей вероятностью колышек опустится с той или другой стороны от него. А поскольку робот не будет иметь информации о том, с какой стороны от отверстия опустился колышек, то не будет знать, куда его двигать дальше.

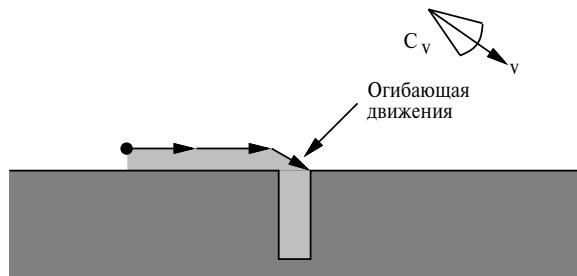


*Рис. 25.16. Двухмерная среда, конус неопределенности скорости и огибающая возможных движений робота. Намеченная скорость равна  $v$ , но из-за неопределенности фактическая скорость может находиться в пределах  $C_v$ , а это приводит к тому, что окончательная конфигурация может определяться в любой точке внутри огибающей движения. Это означает, что возникает неопределенность в отношении того, удастся ли попасть в отверстие или нет*

Более приемлемая стратегия показана на рис. 25.17 и 25.18. На рис. 25.17 приведен пример того, как робот намеренно направляет свое движение в определенную сторону от отверстия. Условия выполнения команды движения показаны на рисунке, а проверкой окончания движения становится контакт с любой поверхностью. На рис. 25.18 показано, что выдается команда движения, которая вынуждает робота передвигать колышек, скользящий по поверхности, до его попадания в отверстие. Тем самым предполагается использование команды приспособляемого движения. Поскольку все возможные скорости в огибающей движения направлены влево, робот должен передвинуть скользящий по поверхности колышек вправо после вступления его в контакт с горизонтальной поверхностью. Вслед за прикосновением колышка с правой вертикальной стенкой отверстия колышек должен проскользнуть вниз, поскольку все возможные скорости направлены вниз относительно этой вертикальной поверхности. Колышек будет продолжать двигаться до тех пор, пока не достигнет дна отверстия, поскольку именно таково условие завершения этого движения. Несмотря на неопределенность управления, все возможные траектории движения робота оканчиваются контактом с дном отверстия, разумеется, при условии, что не обнаружатся какие-либо дефекты поверхности, из-за которых робот не сможет сдвинуть с места застрявший колышек.



*Рис. 25.17. Первая команда движения и полученная в итоге огибающая возможных движений робота. Независимо от любых погрешностей, известно, что в окончательной конфигурации колышек будет находиться слева от отверстия*



*Рис. 25.18. Вторая команда движения и огибающая возможных движений. Даже при наличии погрешностей колышек в конечном итоге оказывается в отверстии*

Вполне очевидно, что задача конструирования планов тонких движений не тривиальна; в действительности она намного сложнее по сравнению с задачей планирования в условиях точных движений. Для решения этой задачи можно либо выбирать постоянное количество дискретных значений для каждого движения, либо использовать геометрию среды для выбора направлений, позволяющих определить качественно иное поведение. Планировщик тонких движений принимает в качестве входных данных описание пространства конфигураций, угол наклона конуса неопределенности скоростей и спецификацию возможных сенсорных восприятий, определяющих ситуацию завершения (в данном случае — контакт с поверхностью). В свою очередь, планировщик должен выработать многоэтапный условный план (или политику), позволяющий добиться гарантированного успеха, если подобный план существует.

В рассматриваемом примере предполагается, что планировщик имеет в своем распоряжении точную модель среды, но возможно также принять допущение о наличии ограниченной погрешности в этой модели, как показано ниже. Если погрешность может быть описана в терминах параметров, то соответствующие параметры добавляются в качестве степеней свободы в пространство конфигураций. В частности, в последнем примере, если бы была неопределенность в отношении глубины и ширины отверстия, то можно было бы добавить эти величины в пространство конфигураций как две степени свободы. Из этого не следует, что появится возмож-

ность передвигать захвата непосредственно в направлении этих степеней свободы в пространстве конфигураций или получать информацию об их позиции. Но оба эти ограничения можно учесть, описывая задачу как задачу FMP, задавая соответствующим образом данные о неопределенностях средств управления и датчиков. В результате возникает сложная, четырехмерная задача планирования, но появляется возможность применять точно такие же методы планирования, как и раньше. Следует отметить, что надежный подход такого рода приводит к созданию планов, в которых учитываются результаты самого неблагоприятного развития событий, а не максимизируется ожидаемое качество плана, в отличие от методов теории решений, описанных в главе 17. Единственным оптимальным аспектом планов действий в наиболее неблагоприятной ситуации (с точки зрения теории решений) является то, что они позволяют предотвратить последствия неудачи во время выполнения плана, намного худшие по сравнению с любыми другими затратами, связанными с его выполнением.

## 25.6. ОСУЩЕСТВЛЕНИЕ ДВИЖЕНИЙ

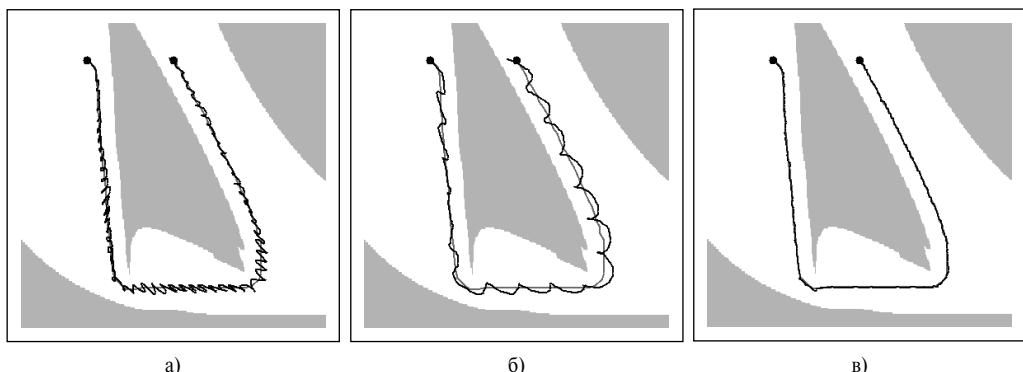
До сих пор речь в данной главе шла о том, как планировать движения, а не как их осуществлять. В разрабатываемых планах (особенно в тех, которые были составлены с помощью детерминированных планировщиков пути) предполагалось, что робот может просто проследовать по любому пути, сформированному алгоритмом. Но в реальном мире, безусловно, дело обстоит иначе. Роботы обладают инерцией и не могут выполнять произвольные команды движения по заданному пути, кроме как на произвольно низких скоростях. В большинстве случаев роботов, выполняя команды движения, прилагает усилия для перемещения в ту или иную точку, а не просто задает нужные ему позиции. В данном разделе описаны методы вычисления таких усилий.

### Динамика и управление

В разделе 25.2 введено понятие **динамического состояния**, которое расширяет представление о кинематическом состоянии робота, позволяя моделировать скорости робота. Например, в описании динамического состояния, кроме данных об угле поворота шарнира робота, отражена скорость изменения этого угла. В модели перехода для любого представления динамического состояния учитываются влияния усилий на эту скорость изменения. Подобные модели обычно выражаются с помощью **дифференциальных уравнений**, которые связывают количество (например, кинематическое состояние) с изменением этого количества во времени (например, скоростью). В принципе, можно было бы выбрать способ планирования движений робота с использованием динамических моделей вместо кинематических моделей, которые рассматривались в предыдущих разделах. Такая методология приводит к достижению превосходных показателей производительности робота, если удается составить нужные планы. Однако динамическое состояние намного сложнее по сравнению с кинематическим пространством, а из-за большого количества измерений задачи планирования движений становятся неразрешимыми для любых роботов, кроме самых простых. По этой причине применяемые на практике робототехнические системы часто основаны на использовании более простых кинематических планировщиков пути.

Общепринятым методом компенсации ограничений кинематических планов является использование для слежения за роботом отдельного механизма, **контроллера**. Контроллерами называются устройства, вырабатывающие команды управления роботом в реальном времени с использованием обратной связи от среды для достижения цели управления. Если цель состоит в удержании робота на заранее запланированном пути, то такие контроллеры часто называют **опорными контроллерами**, а путь называют **опорным путем**. Контроллеры, оптимизирующие глобальную функцию затрат, называют **оптимальными контроллерами**. По существу, оптимальная политика для задачи MDP является определением оптимального контроллера.

На первый взгляд задача управления, позволяющая удерживать робот на заранее заданном пути, кажется относительно простой. Но на практике даже в ходе решения этой внешне простой задачи могут встретиться некоторые ловушки. На рис. 25.19, *a* показано, какие нарушения могут при этом возникать. На данном рисунке демонстрируется путь робота, предпринимающего попытку следовать по кинематическому пути. После возникновения любого отклонения (обусловленного либо шумом, либо ограничениями на те усилия, которые может применять робот) робот прикладывает противодействующее усилие, величина которого пропорциональна этому отклонению. Интуитивные представления говорят о том, что такой подход якобы вполне оправдан, поскольку отклонения должны компенсироваться противодействующим усилием, чтобы робот не отклонялся от своей траектории. Однако, как показано на рис. 25.19, *a*, действия такого контроллера вызывают довольно интенсивную вибрацию робота. Эта вибрация является результатом естественной инерции манипулятора робота — робот, резко направленный в стороны опорной позиции, проскаивает эту позицию, что приводит к возникновению симметричной погрешности с противоположным знаком. Согласно кривой, приведенной на рис. 25.19, *a*, такое перегулирование может продолжаться вдоль всей траектории, поэтому результатирующее движение робота далеко от идеального. Очевидно, что нужно предусмотреть лучший способ управления.



*Рис. 25.19. Управление манипулятором робота с использованием различных методов: пропорциональное управление с коэффициентом усиления 1,0 (а); пропорциональное управление с коэффициентом усиления 0,1 (б); пропорционально-дифференциальное управление с коэффициентами усиления 0,3 для пропорционального и 0,8 для дифференциального компонента (в). Во всех случаях предпринимается попытка провести манипулятор робота по пути, обозначенному серым цветом*

Для того чтобы понять, каким должен быть лучший контроллер, опишем формально тот тип контроллера, который допускает перерегулирование. Контроллеры, прикладывающие усилия, обратно пропорциональные наблюдаемой погрешности, называются **P-контроллерами**. Буква Р является сокращением от *proportional* (пропорциональный) и показывает, что фактическое управляющее воздействие пропорционально погрешности позиционирования манипулятора робота. В качестве более формальной постановки допустим, что  $y(t)$  — опорный путь, параметризованный времененным индексом  $t$ . Управляющее воздействие  $a_t$ , выработанное Р-контроллером, имеет следующую форму:

$$a_t = K_p(y(t) - x_t)$$

где  $x_t$  — состояние робота во время  $t$ ;  $K_p$  — так называемый **коэффициент усиления** контроллера, от которого зависит, какое усилие будет прилагать контроллер, компенсируя отклонения между фактическим состоянием  $x_t$  и желаемым  $y(t)$ . В данном примере  $K_p=1$ . На первый взгляд может показаться, что проблему можно устранить, выбрав меньшее значение для  $K_p$ . Но, к сожалению, дело обстоит иначе. На рис. 25.19, б показана траектория манипулятора робота при  $K_p=1$ , в которой все еще проявляется колебательное поведение. Уменьшение величины коэффициента усиления способствует лишь уменьшению интенсивности колебаний, но не устраивает проблему. В действительности в отсутствие трения Р-контроллер действует в соответствии с законом пружины, поэтому он до бесконечности совершает колебания вокруг заданной целевой точки.

В традиционной науке задачи такого типа принадлежат к области **теории управления**, которая приобретает всю большую важность для исследователей в области искусственного интеллекта. Исследования в этой области, проводившиеся в течение десятков лет, привели к созданию многих типов контроллеров, намного превосходящих описанный выше контроллер, действующий на основании простого закона управления. В частности, опорный контроллер называется **стабильным**, если небольшие возмущения приводят к возникновению ограниченной погрешности, связывающей сигнал робота и опорный сигнал. Контроллер называется **строго стабильным**, если он способен вернуть управляемый им аппарат на опорный путь после воздействия подобных возмущений. Очевидно, что рассматриваемый здесь Р-контроллер только внешне кажется стабильным, но не является строго стабильным, поскольку не способен обеспечить возвращение робота на его опорную траекторию.

Простейший контроллер, позволяющий добиться строгой стабильности в условиях данной задачи, известен под названием **PD-контроллера**. Буква Р снова является сокращением от *proportional* (пропорциональный), а D — от *derivative* (дифференциальный). Для описания PD-контроллеров применяется следующее уравнение:

$$a_t = K_p(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} \quad (25.3)$$

Согласно этому уравнению, PD-контроллеры представляют собой Р-контроллеры, дополненные дифференциальным компонентом, который добавляет к значению управляющего воздействия  $a_t$  терм, пропорциональный первой производной от погрешности  $y(t) - x_t$  по времени. К какому результату приводит добавление такого терма? Вообще говоря, дифференциальный терм гасит колебания в системе, для управле-

ния которой он применяется. Чтобы убедиться в этом, рассмотрим ситуацию, в которой погрешность  $(y(t) - x_t)$  резко изменяется во времени, как было в случае с P-контроллером, описанным выше. При этом производная такой погрешности прикладывается в направлении, противоположном пропорциональному терму, что приводит к уменьшению общего отклика на возмущение. Однако, если та же погрешность продолжит свое присутствие и не изменится, то производная уменьшится до нуля и при выборе управляющего воздействия будет доминировать пропорциональный терм.

Результаты применения такого PD-контроллера для управления манипулятором робота при использовании в качестве коэффициентов усиления значений  $K_p = .3$  и  $K_d = .8$  показаны на рис. 25.19, в. Очевидно, что в конечном итоге траектория манипулятора стала гораздо более гладкой и на ней внешне не заметны какие-либо колебания. Как показывает этот пример, дифференциальный терм позволяет обеспечить стабильность работы контроллера в тех условиях, когда она недостижима другими способами.

Но, как показывает практика, PD-контроллеры также создают предпосылки отказов. В частности, PD-контроллеры могут оказаться неспособными отрегулировать погрешность до нуля, даже при отсутствии внешних возмущений. Такой вывод не следует из приведенного в этом разделе примера робота, но, как оказалось, иногда для уменьшения погрешности до нуля требуется приложить обратную связь с пропорциональным перерегулированием. Решение этой проблемы заключается в том, что к закону управления нужно добавить третий терм, основанный на результатах интегрирования погрешности по времени:

$$a_t = K_p(y(t) - x_t) + K_I \int (y(t) - x_t) dt + K_D \frac{\partial(y(t) - x_t)}{\partial t} \quad (25.4)$$

где  $K_I$  — еще один коэффициент усиления. В терме

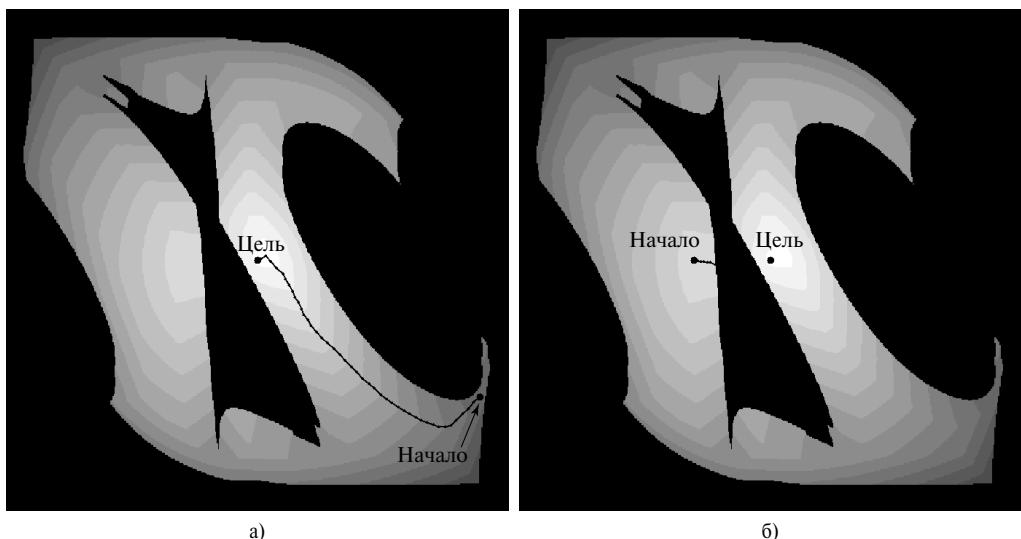
$$\int (y(t) - x_t) dt$$

вычисляется интеграл погрешности по времени. Под влиянием этого терма корректируется продолжающиеся долгое время отклонения между опорным сигналом и фактическим состоянием. Если, например,  $x_t$  меньше чем  $y(t)$  в течение продолжительного периода времени, то значение этого интеграла возрастает до тех пор, пока результатирующее управляющее воздействие  $a_t$  не вызовет уменьшение этой погрешности. Таким образом, интегральные термы гарантируют отсутствие систематических погрешностей в действиях контроллера за счет повышения опасности колебательного поведения. Контроллер, закон управления которого состоит из всех трех термов, называется **PID-контроллером**. PID-контроллеры широко используются в промышленности для решения самых различных задач управления.

## Управление на основе поля потенциалов

Выше в данной главе поле потенциалов было определено как дополнительная функция затрат в планировании движений робота, но поле потенциалов может также использоваться для непосредственной выработки траектории движения робота,

что позволяет полностью отказаться от этапа планирования пути. Для достижения этой цели необходимо определить притягивающее усилие, которое влечет манипулятор робота в направлении его целевой конфигурации, и поле потенциалов, отталкивающее манипулятор, которое отводит манипулятор от препятствий. Такое поле потенциалов показано на рис. 25.20. Его единственным глобальным минимумом является целевая конфигурация, а стоимость измеряется суммой расстояния до целевой конфигурации и дальности до препятствий. Для формирования поля потенциалов, показанного на этом рисунке, не требовалось никакого планирования. Благодаря такой их особенности поля потенциалов могут успешно применяться в управлении в реальном времени. На рис. 25.20 показаны две траектории робота, осуществляющего восхождение к вершине в поле потенциалов при двух различных начальных конфигурациях. Во многих приложениях поле потенциалов может быть эффективно рассчитано для любой конкретной конфигурации. Кроме того, оптимизация потенциала сводится к вычислению градиента потенциала для текущей конфигурации робота. Такие вычисления обычно являются чрезвычайно эффективными, особенно по сравнению с алгоритмами планирования пути, которые связаны с затратами, характеризующими экспоненциальной зависимостью от размерностей пространства конфигураций (от степеней свободы).



*Рис. 25.20. Метод управления на основе поля потенциалов. Траектория робота восходит по градиенту поля потенциалов, состоящего из отталкивающих усилий, обусловленных наличием препятствий, и притягивающих усилий, которые соответствуют целевой конфигурации: успешно проделанный путь (а); локальный оптимум (б)*

Тот факт, что подход на основе поля потенциалов позволяет так эффективно находить путь к цели, даже если при этом приходится преодолевать большие расстояния в пространстве конфигураций, ставит под вопрос саму целесообразность использования планирования в робототехнике. Действительно ли методы на основе поля потенциалов позволяют решать все задачи, или такая ситуация, как в данном примере, является просто результатом благоприятного стечения обстоятельств? Ответ заключается в том, что в данном случае обстоятельства действительно складыва-

лись благоприятно, а в других условиях поля потенциалов имеют много локальных минимумов, которые могут стать ловушкой для робота. В данном примере робот приближается к препятствию, вращая шарнир своего плеча до тех пор, пока не заходит в тупик, оказавшись не с той стороны препятствия, с какой нужно. Поле потенциалов не имеет достаточно развитой конфигурации для того, чтобы вынудить робот согнуть свой локоть, что позволило бы ему пропустить манипулятор под препятствием. Иными словами, методы на основе поля потенциалов превосходно подходят для локального управления роботом, но все еще требуют глобального планирования. Еще одним важным недостатком, связанным с использованием поля потенциалов, является то, что усилия, вырабатываемые при этом подходе, зависят только от положений препятствия и робота, а не от скорости робота. Таким образом, метод управления на основе поля потенциалов в действительности является кинематическим и может оказаться неприменимым для быстро движущегося робота.

## Реактивное управление

До сих пор в этой главе речь шла об управляющих решениях, которые требуют наличия определенной модели среды, чтобы на ее основе можно было сформировать либо опорный путь, либо поле потенциалов. Но с этим подходом связаны некоторые сложности. Во-первых, зачастую сложно получить достаточно точные модели, особенно в сложной или удаленной среде, такой как поверхность Марса. Во-вторых, даже в тех случаях, когда есть возможность составить модель с достаточной точностью, вычислительные сложности и погрешности локализации могут привести к тому, что эти методы окажутся практически не применимыми. В определенных обстоятельствах более подходящим становится один из видов рефлексного проекта агента — проект на основе так называемого **реактивного управления**.

Одним из примеров такого проекта является шестиногий робот, или **гексапод**, показанный на рис. 25.21, *a*, который предназначен для ходьбы по пересеченной местности. В целом датчики робота не позволяют формировать модели местности с точностью, достаточной для любого из методов планирования пути, описанных в предыдущем разделе. Кроме того, даже в случае использования достаточно точных датчиков задача планирования пути не разрешима с помощью имеющихся вычислительных средств из-за наличия двенадцати степеней свободы (по две для каждой ноги).

Тем не менее существует возможность определить спецификацию контроллера непосредственно, без использования явной модели среды. (Выше в данной главе такой подход уже был продемонстрирован на примере PD-контроллера, который оказался способным вести сложный манипулятор робота к цели при отсутствии явной модели динамики робота; однако для этого контроллера требовался опорный путь, сформированный с помощью кинематической модели.) Для рассматриваемого примера шагающего робота после выбора подходящего уровня абстракции задача определения закона управления оказалась удивительно простой. В приемлемом законе управления может быть предусмотрено циклическое движение каждой ноги с тем, чтобы эта нога на какой-то момент касалась земли, а в остальное время двигалась в воздухе. Координация действий всех шести ног должна осуществляться так, чтобы три из них (расположенные на противоположных концах) всегда находились на земле для обеспечения физической опоры. Такой принцип управления можно легко за-программировать, и он себя полностью оправдывает на ровной местности. А на пе-

рессеченной местности движению ног вперед могут помешать препятствия. Это затруднение можно преодолеть с помощью исключительно простого правила управления: если движение какой-то ноги вперед блокируется, следует отвести ее немного назад, поднять выше и предпринять еще одну попытку. Созданный в итоге контроллер показан на рис. 25.21, б в виде конечного автомата; он представляет собой рефлексный агент с поддержкой состояния, в котором внутреннее состояние представлено индексом текущего состояния автомата (от  $s_1$  до  $s_4$ ).

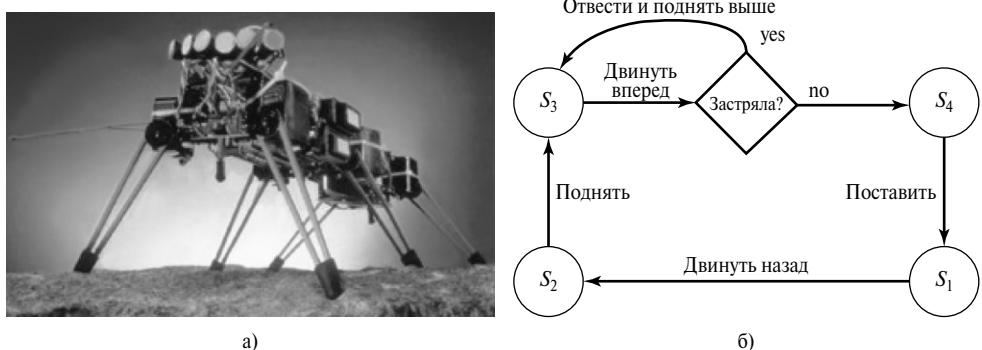


Рис. 25.21. Пример применения реактивного управления: шестиногий робот (а); дополненный конечный автомат (Augmented Finite State Machine — AFSM) для управления одной ногой (б). Обратите внимание на то, что этот автомат AFSM реагирует на сенсорную обратную связь: если какая-то нога не может двигаться вперед при выполнении этапа ее поворота и переноса в прямом направлении, то она поднимается каждый раз все выше и выше

Практика показала, что разновидности такого простого контроллера, действующего на основе обратной связи, позволяют реализовывать исключительно надежные способы ходьбы, с помощью которых робот свободно маневрирует на пересеченной местности. Очевидно, что в таком контроллере не используется модель, кроме того, для выработки управляющих действий не осуществляется алгоритмический вывод и не производится поиск. В процессе эксплуатации подобного контроллера решающую роль в выработке поведения роботом играет обратная связь от среды. Само программное обеспечение робота, отдельно взятое, не определяет, что фактически происходит после того, как робот входит в какую-то среду. Поведение, проявляющееся в результате взаимодействия (простого) контроллера и (сложной) среды, часто называют **эмерджентным поведением** (т.е. поведением не планируемым, а обусловленным ситуацией). Строго говоря, все роботы, рассматриваемые в этой главе, обнаруживают эмерджентное поведение в связи с тем фактом, что ни одна из используемых в них моделей не является идеальной. Но по традиции этот термин применяется для обозначения лишь таких методов управления, в которых не используются явно заданные модели среды. Кроме того, эмерджентное поведение является характерным для значительной части биологических организмов.

С формальной точки зрения реактивные контроллеры представляют собой одну из форм реализации политики для задач MDP (или, если они имеют внутреннее состояние, для задач POMDP). В главе 17 было описано несколько методов выработки политики на основании модели робота и его среды. В робототехнике большое практическое значение имеет подход, предусматривающий составление подобной политики

вручную, поскольку часто невозможно сформулировать точную модель. В главе 21 описаны методы обучения с подкреплением, позволяющие формировать политику на основании опыта. Некоторые из подобных методов (такие как Q-обучение и поиск политики) не требуют модели среды и позволяют создавать высококачественные контроллеры для роботов, но взамен требуют предоставления огромных объемов обучающих данных.

## 25.7. АРХИТЕКТУРЫ РОБОТОТЕХНИЧЕСКОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

❖ **Архитектурой программного обеспечения** называется применяемая методология структуризации алгоритмов. В понятие архитектуры обычно включают языки и инструментальные средства для написания программ, а также общий подход к тому, как должно быть организовано взаимодействие программ.

Предложения по созданию современных архитектур программного обеспечения для робототехники должны отвечать на вопрос о том, как сочетаются реактивные средства управления и основанные на модели алгоритмические средства управления. Преимущества и недостатки реактивных и алгоритмических средств управления таковы, что эти средства во многом дополняют друг друга. Реактивные средства управления действуют на основании сигналов, полученных от датчиков, и хорошо подходят для принятия решений низкого уровня в реальном времени. Однако реактивные средства управления лишь в редких случаях позволяют вырабатывать приемлемые решения на глобальном уровне, поскольку глобальные управляющие решения зависят от информации, которая не может быть воспринята с помощью датчиков во время принятия решения. Для таких задач в большей степени подходят алгоритмические средства управления.

В связи с этим в большинстве вариантов робототехнических архитектур реактивные методы используются на низких уровнях управления, а алгоритмические методы — на более высоких уровнях. Такая комбинация средств управления уже встречалась в данной главе в описании PD-контроллеров, где было показано, как сочетается (реактивный) PD-контроллер с (алгоритмическим) планировщиком пути. Архитектуры, в которых сочетаются реактивные и алгоритмические методы, обычно называют ❖ **гибридными архитектурами**.

### Обобщающая архитектура

❖ **Обобщающая архитектура** [189] представляет собой инфраструктуру для сборки реактивных контроллеров из конечных автоматов. Узлы этих автоматов могут содержать средства проверки для некоторых сенсорных переменных, и в таких случаях трассировка выполнения конечного автомата становится обусловленной результатом подобной проверки. Дуги могут быть размечены сообщениями, которыерабатываются при прохождении по этим дугам, после чего сообщения передаются в двигатели робота или другие конечные автоматы. Кроме того, конечные автоматы оборудованы внутренними таймерами (часами), которые управляют затратами времени, требуемыми для прохождения дуги. Полученные в итоге автоматы обычно на-

зывают **дополненными конечными автоматами**, или сокращенно AFSM (Augmented Finite State Machine), где под дополнением подразумевается использование часов.

Примером простого автомата AFSM может служить автомат с четырьмя состояниями, показанный на рис. 25.21, б, который вырабатывает команды циклического движения ноги для шестиногого шагающего робота. Этот автомат AFSM реализует циклический контроллер, который действует в основном без учета обратной связи от среды. Тем не менее обратная связь от датчика учитывается на этапе переноса ноги вперед. Если нога не может пройти вперед, что означает наличие впереди препятствия, робот отводит ногу немного назад, поднимает ее выше и предпринимает попытку еще раз выполнить шаг вперед. Поэтому данный контроллер приобретает способность реагировать на непредвиденные ситуации, возникающие в процессе взаимодействия робота со своей средой.

В обобщающей архитектуре предусмотрены дополнительные примитивы для синхронизации работы автоматов AFSM, а также для комбинирования выходных данных, поступающих от многочисленных, возможно конфликтующих автоматов AFSM. Благодаря этому такая архитектура позволяет программисту компоновать все более и более сложные контроллеры по принципу восходящего проектирования. В рассматриваемом примере можно начать с создания автоматов AFSM для отдельных ног, а вслед за этим ввести автомат AFSM для координации действий одновременно нескольких ног. В качестве надстройки над этими автоматами можно реализовать такое поведение высокого уровня, как предотвращение столкновений с крупными препятствиями, для чего могут потребоваться такие операции, как отступление и изменение направления движения.

Идея компоновки контроллеров роботов из автоматов AFSM весьма заманчива. Достаточно представить себе, насколько сложно было бы выработать такое же поведение с использованием любого из алгоритмов планирования пути в пространстве конфигураций, описанных в предыдущем разделе. Прежде всего, потребовалось бы точная модель местности. Пространство конфигураций робота с шестью ногами, каждая из которых приводится в действие двумя отдельными двигателями, имеет общее количество измерений, равное восемнадцати (двенадцать измерений для конфигурации ног и шесть для локализации и ориентации робота относительно его среды). Даже если бы применяемые компьютеры были достаточно быстрыми для того, чтобы обеспечить поиск путей в таких многомерных пространствах, все равно нельзя было бы избавиться от необходимости учитывать такие неприятные ситуации, как скольжение робота по склону. Из-за таких стохастических эффектов единственный найденный путь через пространство конфигураций был бы наверняка слишком трудновыполнимым, и с возможными непредвиденными ситуациями не помог бы справиться даже PID-контроллер. Иными словами, задача выработки поведения, определяющего движение с помощью алгоритмических расчетов, слишком сложна для современных алгоритмов планирования движения робота.

К сожалению, обобщающая архитектура также не лишена недостатков. Во-первых, автоматы AFSM обычно действуют под управлением непосредственно полученных сенсорных входных данных. Такая организация работы оправдывает себя, если сенсорные данные надежны и содержат всю необходимую информацию для принятия решения, но становится неприемлемой, если есть необходимость агрегировать сенсорные данные, полученные за определенные промежутки времени, с помощью нетривиальных способов. Поэтому контроллеры обобщающего типа в основном

применяются для решения локальных задач, таких как прохождение вдоль стены или движение к видимым источникам света. Во-вторых, из-за отсутствия средств выполнения алгоритмических расчетов изменение задания для робота становится затруднительным. Робот обобщающего типа обычно выполняет только одно задание, и нет такого способа, который позволил бы модифицировать его средства управления таким образом, чтобы он приспособился к выполнению других задач управления (в этом роботы такого типа полностью аналогичны навозному жуку, о котором речь шла на с. 81). Наконец, в работе контроллеров обобщающего типа трудно разобраться. На практике возникают такие запутанные взаимодействия десятков одновременно функционирующих автоматов AFSM (и со средой), что большинство программистов не в состоянии их проанализировать. По всем этим причинам, несмотря на свою огромную историческую важность, обобщающая архитектура редко используется в коммерческой робототехнике. Но для некоторых ее потомков судьба сложилась иначе.

### Трехуровневая архитектура

В гибридных архитектурах реакции объединяются с алгоритмическими вычислениями. Одним из видов гибридной архитектуры, намного превосходящим другие по своей популярности, является **трехуровневая архитектура**, которая состоит из реактивного, исполнительного и алгоритмического уровней.

➤ **Реактивный уровень** обеспечивает низкоуровневое управление роботом. Отличительной особенностью этого уровня является наличие жесткого цикла “восприятие–действие”. Время принятия решения на этом уровне часто составляет лишь несколько миллисекунд.

➤ **Исполнительный уровень** (или уровень упорядочения) служит в качестве посредника между реактивным и алгоритмическим уровнями. Он принимает директивы от алгоритмического уровня и упорядочивает их для передачи на реактивный уровень. Например, на исполнительном уровне может быть выполнена обработка множества промежуточных пунктов, сформированного алгоритмическим планировщиком пути, а затем приняты решения о том, какое реактивное поведение должно быть вызвано. Время принятия решения на исполнительном уровне обычно составляет порядка одной секунды. Исполнительный уровень отвечает также за интеграцию сенсорной информации в виде представления внутреннего состояния. Например, на этом уровне могут быть реализованы процедуры локализации робота и оперативного составления карты.

На ➤ **алгоритмическом уровне**рабатываются глобальные решения сложных задач с использованием методов планирования. Из-за вычислительной сложности, связанной с выработкой таких решений, время принятия решений на этом уровне составляет порядка нескольких минут. На алгоритмическом уровне (или уровне планирования) для принятия решений используются модели. Эти модели могут быть подготовлены заранее или сформированы путем обучения на основе имеющихся данных, и в них обычно используется информация о состоянии, собранная на исполнительном уровне.

В большинстве современных систем робототехнического программного обеспечения используются те или иные варианты трехуровневой архитектуры. Но критерии декомпозиции на три уровня нельзя назвать очень строгими. К тому же в неко-

торых системах робототехнического программного обеспечения предусмотрены дополнительные уровни, такие как уровни пользовательского интерфейса, которые управляют взаимодействием с пользователями, или уровни, ответственные за координацию действий робота с действиями других роботов, функционирующих в той же среде.

## Робототехнические языки программирования

Многие робототехнические контроллеры реализованы с использованием языков программирования специального назначения. Например, многие программы для обобщающей архитектуры были реализованы на **языке поведения**, который был определен Бруксом [191]. Этот язык представляет собой язык управления в реальном времени на основе правил, результатом компиляции которого становятся контроллеры AFSM. Отдельные правила этого языка, заданные с помощью синтаксиса, подобного Lisp, компилируются в автоматы AFSM, а группы автоматов AFSM объединяются с помощью совокупности механизмов передачи локальных и глобальных сообщений.

Так же как и обобщающая архитектура, язык поведения является ограниченным, поскольку он нацелен на создание простых автоматов AFSM с относительно узким определением потока связи между модулями. Но в последнее время на базе этой идеи проведены новые исследования, которые привели к созданию целого ряда языков программирования, аналогичных по своему духу языку поведения, но более мощных и обеспечивающих более быстрое выполнение. Одним из таких языков является **универсальный робототехнический язык**, или сокращенно GRL (Generic Robot Language) [681]. GRL — это функциональный язык программирования для создания больших модульных систем управления. Как и в языке поведения, в GRL в качестве основных конструктивных блоков используются конечные автоматы. Но в качестве настройки над этими автоматами язык GRL предлагает гораздо более широкий перечень конструкций для определения коммуникационного потока и синхронизации ограничений между различными модулями, чем язык поведения. Программы на языке GRL компилируются в эффективные программы на таких языках команд, как С.

Еще одним важным языком программирования (и связанной с ним архитектурой) для параллельного робототехнического программного обеспечения является система планирования реактивных действий, или сокращенно RAPS (Reactive Action Plan System) [470]. Система RAPS позволяет программистам задавать цели, планы, связанные с этими целями (или частично определять политику), а также задавать условия, при которых эти планы по всей вероятности будут выполнены успешно. Крайне важно то, что в системе RAPS предусмотрены также средства, позволяющие справиться с неизбежными отказами, которые возникают в реальных робототехнических системах. Программист может задавать процедуры обнаружения отказов различных типов и предусматривать процедуру устранения исключительной ситуации для каждого типа отказа. В трехуровневых архитектурах система RAPS часто используется на исполнительном уровне, что позволяет успешно справляться с непредвиденными ситуациями, не требующими перепланирования.

Существует также несколько других языков, которые обеспечивают использование в роботах средств формирования рассуждений и средств обучения. Например, Golog [918] представляет собой язык программирования, позволяющий обеспечить безуказиценное взаимодействие средств алгоритмического решения задач (планирования) и средств реактивного управления, заданных непосредственно с помощью специфика-

ции. Программы на языке Golog формулируются в терминах ситуационного исчисления (см. раздел 10.3) с учетом дополнительной возможности применения операторов недетерминированных действий. Кроме спецификации программы управления с возможностями недетерминированных действий, программист должен также предоставить полную модель робота и его среды. Как только программа управления достигает точки недетерминированного выбора, вызывается планировщик (заданный в форме программы доказательства теорем) для определения того, что делать дальше. Таким образом программист может определять частично заданные контроллеры и опираться на использование встроенных планировщиков для принятия окончательного выбора плана управления. Основной привлекательной особенностью языка Golog является предусмотренная в нем безуказицненная интеграция средств реактивного управления и алгоритмического управления. Несмотря на то что при использовании языка Golog приходится соблюдать строгие требования (полная наблюдаемость, дискретные состояния, полная модель), с помощью этого языка были созданы высокоуровневые средства управления для целого ряда мобильных роботов, предназначенных для применения внутри помещений.

Язык **CES** (сокращение от C++ for embedded systems — C++ для встроенных систем) — это языковое расширение C++, в котором объединяются вероятностные средства и средства обучения [1507]. В число типов данных CES входят распределения вероятностей, что позволяет программисту проводить расчеты с использованием неопределенной информации, не затрачивая тех усилий, которые обычно связаны с реализацией вероятностных методов. Еще более важно то, что язык CES обеспечивает настройку робототехнического программного обеспечения с помощью обучения на основании примеров, во многом аналогично тому, что осуществляется в алгоритмах обучения, описанных в главе 20. Язык CES позволяет программистам оставлять в коде “промежутки”, которые заполняются обучающими функциями; обычно такими промежутками являются дифференцируемые параметрические представления, такие как нейронные сети. В дальнейшем на отдельных этапах обучения, для которых учитель должен задать требуемое выходное поведение, происходит индуктивное обучение с помощью этих функций. Практика показала, что язык CES может успешно применяться в проблемных областях, характерных для частично наблюдаемой и непрерывной среды.

Язык **ALisp** [32] представляет собой расширение языка Lisp. Язык ALisp позволяет программистам задавать недетерминированные точки выбора, аналогичные точкам выбора в языке Golog. Но в языке ALisp для принятия решений применяется не программа доказательства теорем, а средства определения правильного действия с помощью индуктивного обучения, в которых используется обучение с подкреплением. Поэтому язык ALisp может рассматриваться как удобный способ внедрения знаний о проблемной области в процедуру обучения с подкреплением, особенно знаний об иерархической структуре “процедур” желаемого поведения. До сих пор язык ALisp применялся для решения задач робототехники только в имитационных исследованиях, но может стать основой многообещающей методологии создания роботов, способных к обучению в результате взаимодействия со своей средой.

## 25.8. ПРИКЛАДНЫЕ ОБЛАСТИ

В настоящем разделе описаны некоторые из основных областей приложения для робототехнической технологии.

- **Промышленность и сельское хозяйство.** Работы издавна предназначались для использования в тех областях, где требуется тяжелый труд, но трудовые процессы достаточно структурированы, чтобы допускать возможность робототехнической автоматизации. Самым удачным примером может служить сборочная линия, на которой манипуляторы уже давно выполняют такие задачи, как сборка, установка деталей, доставка материалов, сварка и окраска. При решении многих из этих задач роботы оказались более экономически эффективными по сравнению с работниками-людьми.

Для использования на открытых площадках конструктивное исполнение в виде роботов получили многие из тяжелых машин, которые применяются для сбора урожая, подземной проходки или рытья котлованов. Например, в университете Carnegie-Mellon недавно был завершен проект, который показал, что роботы могут использоваться для очистки от старой краски корпусов крупных судов, причем выполняют эту операцию в 50 раз быстрее, чем люди, и оказывают гораздо меньшее вредное воздействие на окружающую среду. Кроме того, было показано, что прототипы автономных роботов-проходчиков работают быстрее и точнее, чем люди, при выполнении задач по транспортировке руды в подземных шахтах. Роботы использовались для составления карт заброшенных шахт и канализационных систем с высокой точностью. Хотя проекты многих из этих систем все еще находятся на этапе разработки их прототипов, наступление той эпохи, когда роботы возьмут на себя основную часть полумеханической работы, которая в настоящее время выполняется людьми, — лишь вопрос времени.

- **Транспортировка.** Области применения робототехнических систем транспортировки являются весьма разнообразными, начиная от автономных вертолетов, которые доставляют объекты в те места, доступ к которым трудно получить иными способами, и заканчивая автоматическими инвалидными колясками, которые перевозят людей, не способных самостоятельно управлять этими колясками, или автономными портальными погрузчиками, которые превосходят самых опытных крановщиков по точности доставки контейнеров с судов на автомобили в грузовых доках. Одним из самых наглядных примеров успешно действующих транспортных роботов, предназначенных для работы внутри помещения, называемых также гоферами (gofer — мальчик на побегушках), является робот Helpmate, показанный на рис. 25.22, а. Такие роботы применяются в десятках больниц для транспортировки пищи и других медицинских материалов. Исследователи разработали робототехнические системы, напоминающие автомобили, которые способны автономно передвигаться по автомагистралям или по бездорожью. В условиях промышленных предприятий автономные транспортные средства в настоящее время стали одним из обычных средств доставки материалов на складах и производственных линиях.

Для эксплуатации многих из этих роботов требуется внести в среду их применения определенные модификации. К числу наиболее распространенных модификаций относятся средства локализации, такие как индуктивные контуры в полу, активные маяки, метки в виде штрих-кода и спутники системы GPS. Поэтому нерешенной задачей в робототехнике является проектирование роботов, способных использовать для своей навигации не искусственные при-

способления, а естественные признаки, особенно в таких условиях, как дальнее океанское плавание, где система GPS недоступна.



a)

б)

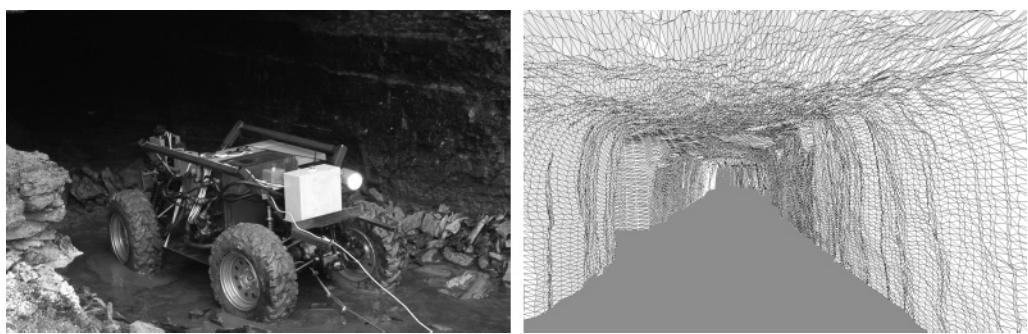
*Рис. 25.22. Примеры применения роботов в медицине: робот Helpmate применяется для транспортировки пищи и других медицинских препаратов в десятках больниц во всем мире (а); хирургические роботы в операционной (предоставлено компанией da Vinci Surgical Systems) (б)*

- **Работа в опасной среде.** Роботы помогали людям при очистке ядерных отходов. К числу наиболее ярких примеров относится устранение последствий аварии в Чернобыле и на острове Тримайл Айленд. Роботы применялись для расчистки завалов Всемирного торгового центра в Нью-Йорке и направлялись на те участки, которые считались слишком опасными для поисковых и спасательных команд.

В некоторых странах роботы используются для транспортировки взрывчатых веществ и обезвреживания бомб; как известно, эти работы характеризуются повышенной опасностью. В настоящее время во многих исследовательских проектах ведется разработка прототипов роботов для обезвреживания минных полей на земле и на море. Большинство существующих роботов для таких задач действуют в режиме телеуправления — ими управляет оператор с помощью средств дистанционного управления. Следующим важным шагом является предоставление таким роботам автономии.

- **Разведка.** Роботы добрались до тех мест, где еще никто не бывал, включая поверхность Марса (см. рис. 25.1, а). С помощью роботов-манипуляторов космонавты выводят на орбиту и снимают с орбиты спутники, а также строят Международную космическую станцию. Кроме того, роботы помогают проводить подводные исследования. Например, с использованием роботов уже было составлено много карт расположения затонувших кораблей. На рис. 25.23 показан робот, составляющий карту заброшенной угольной шахты, наряду с трехмерной моделью карты, формируемой с помощью датчиков расстояния. В 1996 году группа исследователей выпустила шагающего робота в кратер действующего вулкана, чтобы он собрал данные, важные для климатических ис-

следований. В военных операциях широко используются автономные воздушные транспортные средства, известные под названием **беспилотных самолетов**. Роботы становятся очень эффективными средствами сбора информации в тех областях, доступ к которым является сложным (или опасным) для людей.



*Рис. 25.23. Примеры применения роботов в исследованиях труднодоступных участков: робот, используемый для составления карты заброшенной угольной шахты (а); трехмерная карта угольной шахты, составленная роботом (б)*

- **Здравоохранение.** Роботы все чаще используются в качестве помощников хирургов при проведении операций на таких важных органах, как мозг, глаза и сердце. На рис. 25.22, б показана подобная система. Благодаря их высокой точности роботы стали незаменимым инструментальным средством при выполнении некоторых видов операций по эндопротезированию тазобедренного сустава. В экспериментальных исследованиях было обнаружено, что робототехнические устройства снижают опасность повреждений при выполнении колоноскопии. За пределами операционной исследователи стали разрабатывать средства робототехнической помощи для пожилых людей и инвалидов, такие как интеллектуальные автоматизированные шагающие аппараты и интеллектуальные игрушки, которые напоминают о наступлении времени приема лекарства.
- **Персональные услуги.** Перспективной областью применения робототехники становится предоставление услуг. Роботы-ассистенты помогают людям выполнять повседневные задачи. В число коммерчески доступных роботов-помощников по дому входят автономные пылесосы, газонокосилки и подносчики клюшек для гольфа. Все эти роботы способны передвигаться самостоятельно и выполняют свои задачи без помощи людей. Некоторые обслуживающие роботы функционируют в общественных местах; к таким примерам относятся роботизированные справочные бюро, развернутые в крупных университетах и на торговых ярмарках, или автоматические экскурсоводы в музеях. Для успешного выполнения задач обслуживания требуется взаимодействие с людьми и способность надежно реагировать на непредсказуемые и динамические изменения в среде.
- **Развлечения.** Роботы начали завоевывать индустрию игр и развлечений. Выше уже шла речь о роботе AIBO компании Sony (см. рис. 25.4, б); этот игрушечный робот, похожий на собаку, используется в качестве исследовательской

платформы в лабораториях искусственного интеллекта во всем мире. Одной из самых интересных задач искусственного интеллекта, которые изучаются с помощью этой платформы, является **робототехнический футбол**, — соревновательная игра, весьма напоминающая тот футбол, в который играют люди, но проводимая с участием автономных мобильных роботов. Робототехнический футбол открывает широкие возможности для исследований по искусственному интеллекту, поскольку ставит в повестку дня целый ряд задач, способных служить прототипами для многих других, более серьезных робототехнических приложений. Ежегодные робототехнические футбольные соревнования привлекают интерес многих исследователей по искусственному интеллекту и вносят много приятных моментов в эту область робототехники.

- **Дополнение возможностей людей.** Последней прикладной областью робототехнической технологии является дополнение возможностей людей. Например, исследователи разработали шагающие машины для прогулок, которые могут использоваться людьми для передвижения вместо инвалидных колясок. Кроме того, в некоторых исследованиях основные усилия в настоящее время сосредоточены на разработке устройств, которые упрощают для людей задачу ходьбы или передвижения их рук путем приложения дополнительных усилий, действующих через устройства, закрепленные вне скелета. Если крепления таких устройств установлены на постоянной основе, то их можно рассматривать как искусственные роботизированные конечности. Еще одной формой дополнения возможностей людей является роботизированное дистанционное выполнение операций, или так называемое *теле присутствие*. Дистанционное выполнение операций — это осуществление рабочих заданий на больших расстояниях с помощью робототехнических устройств. При выполнении роботизированных дистанционных операций широко применяется такая конфигурация, как “ведущий–ведомый”, в которой робот-манипулятор повторяет движения действующего на удалении оператора-человека, которые передаются через тактильный интерфейс. Все эти системы дополняют возможности человека по взаимодействию с его средой. Некоторые проекты заходят столь далеко, что в них предпринимается попытка создать некое подобие человека, по меньшей мере на очень поверхностном уровне. Например, в настоящее время некоторые японские компании выполняют коммерческие поставки роботов-гуманоидов.

## 25.9. РЕЗЮМЕ

Робототехника — это научно-техническое направление, посвященное созданию интеллектуальных агентов, которые выполняют свои манипуляции в физическом мире. В данной главе представлены перечисленные ниже основные сведения об аппаратном и программном обеспечении роботов.

- Роботы оснащены датчиками для получения результатов восприятия из своей среды и исполнительными механизмами, с помощью которых они могут прилагать физические усилия в своей среде. Большинство роботов представляют

собой либо манипуляторы, закрепленные в фиксированных позициях, либо мобильные роботы, способные передвигаться.

- Робототехническое восприятие предназначено для оценки количественных значений, необходимых для принятия решений, на основании сенсорных данных. Для выполнения этих функций требуются внутреннее представление и метод обновления этого внутреннего представления во времени. В число широко известных сложных задач восприятия входят локализация и составление карты.
- В процессе робототехнического восприятия могут применяться такие вероятностные алгоритмы фильтрации, как фильтры Калмана и фильтры частиц. Эти методы обеспечивают поддержку доверительного состояния, т.е. распределение апостериорных вероятностей по переменным состояния.
- Планирование движений робота обычно осуществляется в пространстве конфигураций, каждая точка которого определяет местонахождение и ориентацию робота, а также углы поворота его шарниров.
- В состав алгоритмов поиска в пространстве конфигураций входят методы декомпозиции ячеек, которые предусматривают декомпозицию пространства всех конфигураций на бесконечное количество ячеек, и методы скелетирования, предусматривающие создание проекций пространства конфигураций на пространства с меньшими размерностями. После этого задача планирования движений решается с использованием поиска в этих более простых структурах.
- Задачу прохождения по пути, найденному с помощью алгоритма поиска, можно решить, используя этот путь в качестве опорной траектории для PID-контроллеров.
- При использовании методов на основе поля потенциалов навигация роботов осуществляется с применением функций потенциалов, параметрами которых служат расстояния до препятствий и целевое местонахождение. Недостатком методов на основе поля потенциалов является то, что при их использовании могут возникать тупиковые ситуации, при которых робот не может выйти из локального минимума. Тем не менее эти методы позволяют непосредственно вырабатывать команды на осуществление движения, не требуя планирования пути.
- Иногда проще непосредственно задать спецификацию контроллера робота, а не определять алгоритмическим способом путь с помощью явно заданной модели среды. Такие контроллеры часто могут быть выполнены в виде простых конечных автоматов.
- Обобщающая архитектура позволяет программистам собирать контроллеры роботов из взаимосвязанных конечных автоматов, дополненныхстроенными таймерами.
- Для разработки робототехнического программного обеспечения, в котором объединяются средства алгоритмического вывода, упорядочения подцелей и управления, широко применяется инфраструктура, основанная на трехуровневой архитектуре.
- Для упрощения разработки программного обеспечения роботов используются языки программирования специального назначения. В этих языках предусмотрены конструкции, предназначенные для разработки многопотокового

программного обеспечения, для интеграции директив управления в средства планирования и для обучения на основе опыта.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Слово **робот** получило широкую известность после выхода в 1921 году пьесы R.U.R. (Rossum's Universal Robots — универсальные роботы Россума) чешского драматурга Карела Чапека. В этой пьесе роботы, которые выращивались из химических растворов, а не конструировались механически, в конечном итоге взбунтовались против своих создателей и решили взять над ними верх. Считается [562], что это слово фактически придумал брат Карела Чапека, Йозеф, который скомбинировал чешские слова “robota” (принудительный труд) и “robotnik” (раб), получив слово “robot”, которое он применил в своем рассказе Opilec, изданном в 1917 году.

Термин “робототехника” (robotics) был впервые использован Айзеком Азимовым [46]. Но само понятие робототехники (которое было известно под многими другими названиями) имеет гораздо более долгую историю. В древнегреческой мифологии упоминается механический человек по имени Талос, который был спроектирован и построен греческим богом огня и кузнечного дела Гефестом. В XVIII столетии разрабатывались восхитительные автоматы (одним из первых примеров таких автоматов является механическая утка Жака Вокансона, созданная в 1738 году), но сложное поведение, которое они демонстрировали, было полностью задано заранее. Возможно, одним из примеров программируемого устройства, подобного роботу, был ткацкий станок Жаккарда (1805 год), описанный в главе 1.

Первым коммерчески поставляемым роботом был робот-манипулятор, получивший название **Unimate** (сокращение от universal automation — универсальная автоматизация). Робот Unimate был разработан Джозефом Энджелбергером и Джорджем Деволом. В 1961 году первый робот Unimate был продан компании General Motors, в которой он использовался при изготовлении телевизионных трубок. Тот же 1961 год примечателен и тем, что в этом году Девол получил первый патент США на конструкцию робота. Через одиннадцать лет, в 1972 году, корпорация Nissan впервые автоматизировала весь сборочный конвейер с помощью роботов, разработанных компанией Kawasaki на основе роботов, поставляемых компанией Unimation Энджелбергера и Девола. Эта разработка стала началом важной научно-технической революции, которая вначале происходила в основном в Японии и США и до сих пор продолжается во всем мире. Следующий шаг был сделан компанией Unimation в 1978 году, когда ее сотрудниками был разработан робот **PUMA** (сокращение от Programmable Universal Machine for Assembly — программируемая универсальная машина для сборки). Робот PUMA, первоначально разработанный для компании General Motors, стал фактически стандартом робототехнических манипуляторов на два следующих десятилетия. В настоящее время количество действующих роботов во всем мире оценивается в один миллион, причем больше половины из них установлено в Японии.

Литературу по робототехническим исследованиям можно разделить приблизительно на две части: мобильные роботы и стационарные манипуляторы. Первым автономным мобильным роботом можно считать “черепаху” Грэя Уолтера, которая была создана в 1948 году, хотя ее система управления не была программируемой.

Робот “Зверь Хопкинса” (Hopkins Beast), созданный в начале 1960-х годов в Университете Джона Хопкинса, был гораздо более сложным; он имел аппаратные средства распознавания образов и был способен обнаружить крышку стандартной розетки питания переменным током. Этот робот обладал способностью находить розетки, подключаться к ним и перезаряжать свои аккумуляторы! Тем не менее “Зверь” имел ограниченный набор навыков. Первым мобильным роботом общего назначения был “Shakey”, разработанный в конце 1990-х годов в институте, называвшемся тогда Станфордским научно-исследовательским институтом (Stanford Research Institute; теперь он носит название SRI) [466], [1143]. “Shakey” был первым роботом, в котором объединялись функции восприятия, планирования и выполнения, и это замечательное достижение оказало влияние на многие дальнейшие исследования по искусственному интеллекту. К другим важным проектам относятся Stanford Cart и CMU Rover [1082]. В [303] описано классическое исследование по автономным транспортным средствам.

В основе развития области робототехнической картографии лежат два различных источника. Первое направление начиналось с работы Смита и Чизмена [1440], которые применяли фильтры Калмана для одновременного решения задач локализации и составления карты. Разработанный ими алгоритм был впервые реализован в исследовании, описанном в [1095], а в дальнейшем дополнен в [912]. В [400] описано современное состояние дел в этой области. Второе направление началось с разработки представления в виде **сетки занятости** для вероятностной картографии, в которой задается вероятность того, что каждый участок  $(x, y)$  занят некоторым препятствием [1083]. Обзор современного состояния дел в робототехнической картографии можно найти в [1508]. Одной из первых работ, в которой было предложено использовать топологическую, а не метрическую картографию, стимулом для которой послужили модели пространственного познания человека, была [863].

Обзор самых ранних методов локализации мобильных роботов приведен в [153]. В теории управления фильтры Калмана применяются в качестве метода локализации в течение многих десятилетий, а в литературе по искусственному интеллекту общая вероятностная формулировка задачи локализации появилась гораздо позже, благодаря работам Тома Дина и его коллег [356], [362], а также Симонса и Кёнига [1412]. В последней работе был предложен термин **марковская локализация**. Первым практическим приложением этого метода явилась работа Бургарда и др. [208], которые создали целый ряд роботов, предназначенных для использования в музеях. Метод локализации Монте-Карло, основанный на фильтрах частиц, был разработан Фоксом и др. [488] и в настоящее время получил широкое распространение. В **фильтре частиц, действующем по принципу Рао-Блэквела**, объединяются средства фильтрации частиц для локализации робота со средствами точной фильтрации для составления карты [1074], [1106].

В течение последнего десятилетия общим стимулом для проведения исследований по мобильным роботам служат два важных соревнования. Ежегодное соревнование мобильных роботов общества AAAI впервые было проведено в 1992 году. Первым победителем соревнования стал робот Carmel [287]. Наблюдаемый прогресс остается стабильным и весьма впечатляющим: в одном из последних соревнований (2002 год) перед роботами стояла задача войти в комплекс зданий, где проводилась конференция, найти путь к бюро регистрации, зарегистрироваться на конференции и произнести речь. В соревновании **RoboCup**, инициатором проведения которого в 1995 году стал Китано со своими коллегами [802], провозглашена цель — к 2050 году

“разработать команду полностью автономных роботов-гуманоидов, которые смогут победить команду чемпионов мира по футболу среди людей”. Игры происходят в лигах для роботов, имитирующих людей, колесных роботов различных размеров и четырехногих роботов Aibo компании Sony. В 2002 году это соревнование привлекло команды почти из 30 различных стран и на нем присутствовало больше 100 тысяч зрителей.

Исследования по созданию роботов-манипуляторов, которые первоначально именовались **машины “рука-глаз”**, развиваются в нескольких весьма различных направлениях. Первой важной работой по созданию машины “рука-глаз” был проект Генриха Эрнста под названием МН-1, описанный в тезисах его докторской диссертации, которые были опубликованы в Массачусетском технологическом институте [441]. Проект Machine Intelligence, разработанный в Эдинбурге, также стал одной из первых демонстраций впечатляющей сборочной системы, основанной на использовании средств машинного зрения, которая получила название Freddy [1044]. После того как был сделан этот решающий начальный вклад, основные усилия были сосредоточены на создании геометрических алгоритмов для решения задач планирования движения в детерминированной и полностью наблюдаемой среде. В оригинальной работе Рейфа [1274] было показано, что задача планирования движения робота является PSPACE-трудной. Представление на основе пространства конфигураций было предложено Лозано-Пересом [956]. Важное влияние оказал ряд статей Шварца и Шарира, посвященных задачам, которые были ими названы задачами для **грузчиков-тяжеловесов** (piano mover) [1371].

Метод рекурсивной декомпозиции ячеек для планирования в пространстве конфигураций был впервые предложен в [193] и существенно доработан в [1646]. Самые первые алгоритмы скелетирования были основаны на диаграммах Вороного [1313] и **графах видимости** [1581]. В [603] приведены результаты разработки эффективных методов инкрементного вычисления диаграмм Вороного, а в [253] диаграммы Вороного были распространены на гораздо более широкий класс задач планирования движения. В тезисах докторской диссертации Джона Кэнни [219] предложен первый полностью экспоненциальный алгоритм планирования движения с использованием еще одного метода скелетирования, называемого алгоритмом  **силуэта**. В книге Жан-Клода Латомба [891] рассматривается целый ряд подходов к решению задачи планирования движения. В [780] описаны методы разработки вероятностных дорожных карт, которые в настоящее время относятся к числу наиболее эффективных методов. Планирование тонких движений с ограниченными сенсорными данными исследовалось в [217] и [957] на основе идеи интервальной неопределенности, а не вероятностной неопределенности. В навигации на основе отметок [902] используются во многом такие же идеи, как и в других областях исследования мобильных роботов.

Управлению роботами как динамическими системами (для решения задач манипулирования или навигации) посвящен огромный объем литературы, поэтому в данной главе лишь кратко рассматриваются сведения, касающиеся этой области. К числу важнейших работ относятся трехтомник по импедансному управлению Хогана [668] и общее исследование по динамике роботов Физерстоуна [456]. Дин и Уэллман [363] были в числе первых, кто попытался связать воедино теорию управления и системы планирования на основе искусственного интеллекта. Тремя классическими учебниками по математическим основам робототехнического манипулирования являются [305], [1184] и [1634]. Важное значение в робототехнике имеет также область проблем **схватывания**, поскольку тема определения стабильных ус-

ловий приложения захвата является весьма сложной [998]. Для того чтобы иметь возможность создать успешно действующее средство схватывания, необходимо обеспечить восприятие данных о прикосновении, или **тактильную обратную связь**, для определения усилия контакта и обнаружения проскальзывания [455].

Понятие управления на основе поля потенциалов, с помощью которого предпринимаются попытки одновременно решать задачи планирования движения и управления, было введено в сферу робототехники Хатибом [793]. В технологии мобильных роботов эта идея стала рассматриваться как практический способ решения задачи предотвращения столкновений, а в дальнейшем была дополнена и легла в основу алгоритма **гистограмм векторного поля** [154]. Функции навигации, представляющие собой робототехническую версию политики управления для детерминированных задач MDP, были предложены в [812].

Вокруг проблемы создания архитектур программного обеспечения для роботов ведутся оживленные дискуссии. Истоками трехуровневой архитектуры стали исследования, проводившиеся еще в эпоху так называемого “старого доброго искусственного интеллекта”, в том числе исследования по созданию проекта Shakey; общий обзор на эту тему приведен в [525]. Обобщающая архитектура была предложена Родни Бруксом [189], хотя аналогичные идеи были разработаны независимо Брейтенбергом [172], в книге которого, *Vehicles*, описан ряд простых роботов, основанных на поведенческом подходе. Вслед за успешным созданием Бруксом шестиногого шагающего робота был разработан целый ряд других проектов. Коннелл описал в своих тезисах докторской диссертации [288] результаты разработки мобильного робота, способного находить объекты, проект которого был полностью реактивным. Описания попыток распространить поведенческий принцип на системы, состоящие из нескольких роботов, можно найти в [999] и [1176]. Результатом обобщения идей робототехники, основанной на параллельно организуемом поведении, стало создание универсальных языков управления роботами GRL [681] и Colbert [832]. В [38] приведен обзор состояния дел в этой области.

Для управления мобильными роботами, выполняющими задания по разведке местности и доставке материалов, использовались также **ситуационные автоматы** [760], [1308], описанные в главе 7. Проекты ситуационных автоматов тесно связаны с проектами, основанными на организации поведения, поскольку они состоят из конечных автоматов, отслеживающих различные аспекты состояния среды с использованием простых комбинаторных схем. Тем не менее в подходе, основанном на организации поведения, подчеркивается отсутствие явного представления, а ситуационные автоматы конструируются алгоритмически из декларативных моделей среды таким образом, что представительное содержание каждого регистра состояния является полностью определенным.

В настоящее время имеется несколько хороших современных учебников по мобильной робототехнике. Кроме учебников, указанных выше, можно назвать сборник Кортенкампа и др. [846], в котором приведен исчерпывающий обзор современных архитектур систем мобильных роботов. В недавно изданных учебниках [423] и [1107] рассматриваются более общие вопросы робототехники. Еще в одной недавно изданной книге по робототехническому манипулированию рассматриваются такие сложные темы, как согласующее движение [997]. Основной конференцией по робототехнике является *IEEE International Conference on Robotics and Automation*. В число робо-

тотехнических журналов входят *IEEE Robotics and Automation*, *International Journal of Robotics Research* и *Robotics and Autonomous Systems*.

## УПРАЖНЕНИЯ

- 25.1.** При любом конечном размере выборки результаты применения алгоритма локализации Монте-Карло являются смещенными (т.е. ожидаемое значение данных о местонахождении, вычисленные с помощью алгоритма, отличаются от истинного ожидаемого значения), поскольку именно так действуют алгоритм фильтрации частиц. В данном упражнении предлагается оценить это смещение.

Для упрощения рассмотрим мир с четырьмя возможными местонахождениями робота:  $x = \{x_1, x_2, x_3, x_4\}$ . Первоначально осуществим равномерную выборку  $N \geq 1$  образцов среди этих местонахождений. Как обычно, вполне приемлемо, если для любого из местонахождений  $x$  будет сформировано больше одной выборки. Допустим, что  $z$  — булева сенсорная переменная, характеризующаяся следующими условными вероятностями:

$$\begin{array}{ll} P(z|x_1) = 0.8 & P(\neg z|x_1) = 0.2 \\ P(z|x_2) = 0.4 & P(\neg z|x_2) = 0.6 \\ P(z|x_3) = 0.1 & P(\neg z|x_3) = 0.9 \\ P(z|x_4) = 0.1 & P(\neg z|x_4) = 0.9 \end{array}$$

В алгоритме MCL эти вероятности используются для формирования весов частиц, которые в дальнейшем нормализуются и используются в процессе повторной выборки. Для упрощения предположим, что при повторной выборке вырабатывается только один новый образец, независимо от  $N$ . Этот образец может соответствовать любому из четырех местонахождений  $x$ . Таким образом, процесс выборки определяет распределение вероятностей по  $X$ .

- a) Каково результирующее распределение вероятностей по  $X$  для этого нового образца? Ответьте на этот вопрос отдельно для  $N=1, \dots, 10$  и для  $N=\infty$ .
- b) Разница между двумя распределениями вероятностей  $P$  и  $Q$  может быть измерена с помощью дивергенции  $KL$ , которая определяется следующим образом:

$$KL(P, Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

Каковы значения дивергенции  $KL$  между распределениями в упр. 25.1, *a* и истинным распределением апостериорных вероятностей?

- b) Какая модификация формулировки задачи (а не алгоритма!) гарантировала бы, чтобы конкретное приведенное выше выражение для оценки оставалось несмешенным даже при конечных значениях  $N$ ? Предложите по меньшей мере две такие модификации (но каждая из них должна соответствовать предложенному заданию).

- 25.2.** Реализуйте алгоритм локализации Монте-Карло для моделируемого робота с датчиками расстояний. Карту, нанесенную на сетку, и данные о расстояниях можно найти в репозитарии кода по адресу [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu). Упражнение будет считаться выполненным, если вы продемонстрируете успешную глобальную локализацию робота.
- 25.3.** Рассмотрим робот-манипулятор, показанный на рис. 25.11. Предположим, что элемент у основания робота имеет длину 60 см, а плечо и предплечье имеют длину по 40 см. Как было указано на с. 1209, обратная кинематика робота часто является не уникальной. Сформулируйте явное решение в замкнутой форме для обратной кинематики этого манипулятора. При каких именно условиях это решение является уникальным?
- 25.4.** Реализуйте алгоритм вычисления диаграммы Вороного для произвольной двухмерной среды, описанной с помощью булева массива с размерами  $n \times n$ . Проиллюстрируйте работу вашего алгоритма, построив график диаграммы Вороного для 10 интересных карт. Какова сложность вашего алгоритма?
- 25.5.** В этом упражнении рассматривается связь между рабочим пространством и пространством конфигураций с использованием примеров, показанных на рис. 25.24.

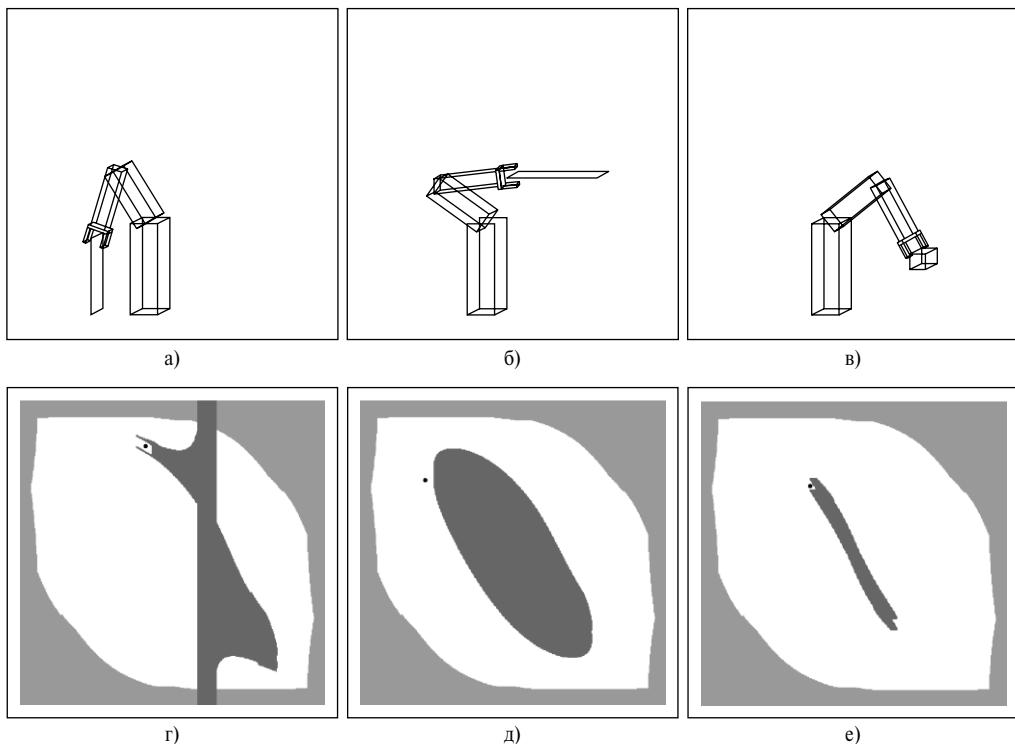


Рис. 25.24. Схемы для упр. 25.5

- a)** Рассмотрим конфигурации роботов, показанных на рис. 25.24, *a–e*, игнорируя препятствие, приведенное на каждом из этих рисунков. Нарисуйте

соответствующие конфигурации манипулятора в пространстве конфигураций. (*Подсказка.* Каждая конфигурация манипулятора отображается на единственную точку в пространстве конфигураций, как показано на рис. 25.11, б.)

- 6) Нарисуйте пространство конфигураций для каждой из диаграмм рабочего пространства, приведенных на рис. 25.24, а–в. (*Подсказка.* В этих пространствах конфигураций общим с пространством конфигураций, показанным на рис. 25.24, а, является тот участок, который соответствует столкновению манипулятора робота с самим собой, а различия обусловлены отсутствием окружающих препятствий и изменением местонахождений препятствий на этих отдельных рисунках.)
  - б) Для каждой из черных точек, приведенных на рис. 25.24, д, е, нарисуйте соответствующие конфигурации манипулятора робота в рабочем пространстве. В этом упражнении не рассматривайте затененные участки.
  - г) Все пространства конфигураций, показанные на рис. 25.24, д, е, сформированы с учетом единственного препятствия в рабочем пространстве (темное затенение), а также ограничений, обусловленных ограничением, препятствующим столкновению манипулятора с самим собой (светлое затенение). Для каждой диаграммы нарисуйте препятствие в рабочем пространстве, которое соответствует участку с темным затенением.
  - д) На рис. 25.24, г, показано, что единственное плоское препятствие способно разбить рабочее пространство на два несвязанных между собой участка. Каково максимальное количество несвязанных участков, которые могут быть созданы в результате вставки плоского препятствия в свободное от препятствий связное рабочее пространство для робота с двумя степенями свободы? Приведите пример и объясните, почему не может быть создано большее количество несвязных участков. Относится ли это утверждение к неплоскому препятствию?
- 25.6. Рассмотрим упрощенный робот, показанный на рис. 25.25. Предположим, что декартовы координаты робота всегда известны, а также известны координаты его целевого местонахождения. Тем не менее неизвестны местонахождения препятствий. Робот, как показано на этом рисунке, может обнаруживать препятствия, находящиеся в непосредственной близости от него. Для упрощения предположим, что движения робота не подвержены шуму и что пространство состояний является дискретным. На рис. 25.25 приведен только один пример; в этом упражнении требуется найти решение для всех возможных миров, заданных в координатной сетке, где действительно имеется путь от начала до целевого местонахождения.
  - а) Спроектируйте алгоритмический контроллер, который гарантирует, что робот всегда достигнет своего целевого местонахождения, если это вообще возможно. Этот алгоритмический контроллер может запоминать в форме карты результаты измерений, которые он получает по мере передвижения робота. Между отдельными операциями перемещения робот может затрачивать произвольно долгое время на алгоритмическую обработку информации.

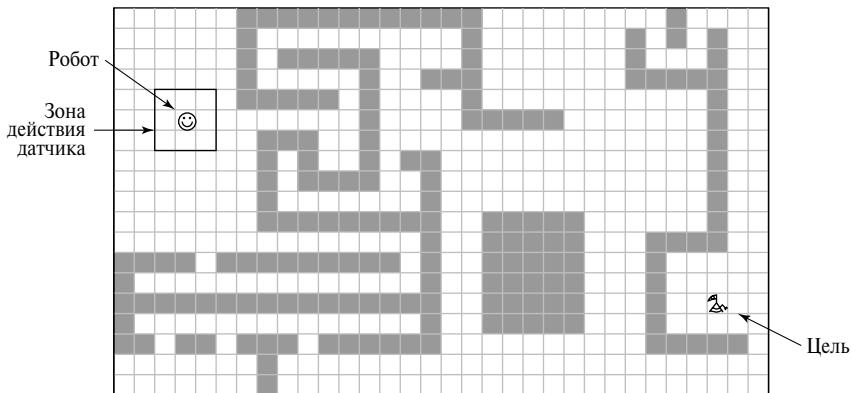


Рис. 25.25. Упрощенный робот в лабиринте (см. упр. 25.6)

- 6)** Теперь спроектируйте реактивный контроллер для выполнения той же задачи. Этот контроллер может не запоминать полученные ранее результаты сенсорных измерений. (Он может даже не составлять карту!) Вместо этого он должен принимать все решения на основании текущих результатов измерений, которые включают данные о его собственном местонахождении и о местонахождении цели. Время, необходимое для выработки решения, не должно зависеть от размеров среды или от количества ранее выполненных шагов. Каково максимальное количество шагов, которые могут потребоваться роботу для достижения цели?
- в)** Какую производительность покажут контроллеры, созданные по условиям упр. 25.6, *a* и *б*, если учитываются только следующие шесть условий: непрерывное пространство состояний, зашумленные результаты восприятия, зашумленные результаты движения, зашумленные результаты восприятия, и движения, неизвестное местонахождение цели (цель может быть обнаружена только в пределах действия датчика расстояний) или движущиеся препятствия. Для каждого условия и каждого контроллера приведите пример ситуации, в которой робот не достигает цели (или объясните, почему такая ситуация не может возникнуть).
- 25.7.** На рис. 25.21, *б* показан дополненный конечный автомат для управления одной ногой шестиногого робота. Цель этого упражнения состоит в том, чтобы спроектировать автомат AFSM, который объединяет шесть экземпляров отдельных контроллеров ног и обеспечивает эффективное, стабильное движение. Для этой цели вы должны дополнить контроллер отдельной ноги так, чтобы он передавал сообщения вновь разрабатываемому автомату AFSM и ожидал поступления других сообщений. Докажите, почему предложенный вами контроллер является эффективным, в том отношении, что он не затрачивает энергию непроизводительно (например, на приведение в движение ног, проскальзывающих в каком-то месте) и что он продвигает робота вперед с достаточно высокой скоростью. Докажите, что предложенный вами контроллер удовлетворяет условию стабильности, приведенному на с. 1194.
- 25.8.** (Это упражнение было впервые предложено Майклом Генезеретом (Michael Genesereth) и Нильсом Нильссоном (Nils Nilsson). В нем могут участвовать

и студенты-первокурсники, и аспиранты.) Люди так успешно выполняют простейшие задачи даже без помощи пальцев, например берут со стола чашки или укладывают кубики в столбики, что часто не осознают, насколько сложными являются эти задачи. Данное упражнение позволяет раскрыть всю сложность элементарных задач и снова пройти путь развития робототехники за последние 30 лет. Вначале нужно выбрать задачу, такую как составление арки из трех блоков. Затем необходимо организовать работу робота с использованием четырех людей, как описано ниже.

- Мозг. Задача Мозга состоит в том, что он должен составлять план достижения цели и управлять руками при выполнении этого плана. Мозг получает входные данные от Глаз, но не может видеть непосредственно саму сцену. Мозг является единственным, кто знает, в чем состоит цель.
- Глаза. Задача Глаз состоит в том, чтобы сообщать Мозгу краткое описание сцены. Глаза должны находиться на расстоянии одного-двух метров от рабочей среды и могут предоставлять ее качественное описание (например, “красная коробка стоит на зеленой коробке, лежащей на боку”) или количественное описание (“зеленая коробка находится слева от синего цилиндра, на расстоянии около полуметра”). Глаза могут также отвечать примерно на такие вопросы Мозга: “Есть ли промежуток между Левой Рукой и красной коробкой?” Если в вашем распоряжении имеется видеокамера, направьте ее на сцену и разрешите Глазам смотреть в видеоискатель видеокамеры, а не прямо на сцену.
- Левая Рука и Правая Рука. Роль каждой Руки играют по одному человеку. Две Руки стоят рядом друг с другом; Левая Рука использует только свою левую руку, а Правая Рука — только свою правую руку. Руки выполняют лишь простые команды от Мозга, например: “Левая Рука, передвинься на пять сантиметров вперед”. Руки не могут выполнять команды, отличные от движений; например: “Подними коробку” — это не та команда, которую может выполнить Рука. Для предотвращения попыток действовать пальцами можно предусмотреть, чтобы Руки носили рукавицы или действовали с помощью клещей. Глаза у Рук должны быть завязаны. Единственные сенсорные возможности, которые им предоставлены, таковы: они имеют право сообщить о том, что путь их движения заблокирован неподвижным препятствием, таким как стол или другая Рука. В подобных случаях Руки могут лишь подать звуковой сигнал, чтобы сообщить Мозгу о возникшем затруднении.



## Часть VIII

### ЗАКЛЮЧЕНИЕ

Философские основания	1248
Настоящее и будущее искусственного интеллекта	1259

## 26 ФИЛОСОФСКИЕ ОСНОВАНИЯ

*В данной главе речь идет о том, что означает “мыслить”, а также о том, могут ли и должны ли этим заниматься искусственно созданные объекты.*

Как упоминалось в главе 1, философы размышляли над мировыми проблемами задолго до того, как появились компьютеры, и пытались решить некоторые проблемы, которые по сути относятся к искусственному интеллекту: “Как функционирует разум? Возможно ли, чтобы машины действовали столь же интеллектуально, как люди, а если ответ на этот вопрос является положительным, то будет ли это означать, что они обладают разумом? Каковы этические последствия создания интеллектуальных машин?” Во всех предыдущих главах настоящей книги рассматривались вопросы, касающиеся самого искусственного интеллекта, а здесь мы в рамках одной главы рассмотрим указанные выше философские проблемы.

Прежде всего введем некоторую терминологию: утверждение, согласно которому машины, возможно, обладают способностью действовать интеллектуально (по-видимому, эту мысль лучше выразить таким образом, что машины, возможно, способны действовать так, как будто действительно являются интеллектуальными), философы называют гипотезой **слабого искусственного интеллекта**, а утверждение, что машины действительно мыслят (а не просто имитируют мыслительные процессы), называется гипотезой **сильного искусственного интеллекта**.

Большинство исследователей искусственного интеллекта принимают гипотезу слабого искусственного интеллекта как данную и не задумываются над тем, что может рассматриваться также гипотеза сильного искусственного интеллекта; коль скоро разработанная ими программа успешно функционирует, их не волнует, назовут ли ее работу имитацией интеллекта или настоящим интеллектом. Но всех исследователей искусственного интеллекта должны заботить этические последствия их деятельности.

## 26.1. СЛАБЫЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ: МОГУТ ЛИ МАШИНЫ ДЕЙСТВОВАТЬ ИНТЕЛЛЕКТУАЛЬНО?

Некоторые философы пытались доказать, что создать искусственный интеллект невозможно, т.е. что машины никогда не смогут действовать интеллектуально. Некоторые из них даже использовали свою эрудицию, чтобы призвать всех прекратить исследования искусственного интеллекта, приводя следующие доводы.

Искусственный интеллект, создаваемый в рамках культа компьютероцентризма, не дает даже ни малейшего шанса на то, что с его помощью удастся добиться долговременных результатов ...настало время направить усилия исследователей искусственного интеллекта (и значительные средства, выделяемые на их поддержку) в области, отличные от этого компьютеризированного подхода [1355].

Очевидно, что ответ на вопрос о том, возможно или невозможно создание искусственного интеллекта, зависит от того, как определено само понятие искусственного интеллекта. По существу создание искусственного интеллекта — это борьба за разработку наилучшей возможной программы агента в данной конкретной архитектуре. При использовании такой формулировки создание искусственного интеллекта возможно по определению, поскольку для любой цифровой архитектуры, состоящей из  $k$  битов памяти, существует точно  $2^k$  программ агентов, и для того чтобы найти наилучшую из них, достаточно просто последовательно проверить их все. Такой подход может оказаться неосуществимым при больших значениях  $k$ , но философы оперируют с теоретическими, а не практическими конструкциями.

Приведенное выше определение искусственного интеллекта вполне подходит для решения технической проблемы поиска приемлемой программы агента при наличии некоторой заданной архитектуры. Поэтому вполне можно было бы закончить этот раздел прямо сейчас, ответив положительно на вопрос, приведенный в его названии. Но философов интересует также проблема сравнения двух вычислительных архитектур — человека и машины. К тому же в философии существует давняя традиция поиска ответа на вопрос: **Могут ли машины мыслить?** К сожалению, сам этот вопрос определен неправильно. Для того чтобы понять, почему это так, рассмотрим приведенные ниже вопросы.

- Могут ли машины совершать полеты?
- Могут ли машины заниматься плаванием?

Большинство людей согласятся, что ответ на первый вопрос является положительным, поскольку самолеты предназначены именно для того, чтобы совершать полеты, но ответят на второй вопрос отрицательно; корабли и подводные лодки движутся над водой и под водой, но это мы не называем *занятиями плаванием*. Тем не менее ни приведенные вопросы, ни ответы не оказывают никакого влияния на повседневную деятельность специалистов по аэронавтике и судоходству, а также на пользователей тех машин, которыми они управляют. К тому же ответы на эти вопросы никак не могут помочь лучше проектировать или расширять возможности самолетов и подводных лодок и в большей степени касаются выбора правильных способов словоупотребления. Слово “заниматься плаванием” (*swim*) в английском языке постепенно приняло смысл “продвигаться в воде за счет движений частей тела”, а слово “совершать полеты” (*fly*) не налагает таких ограничений на выбор способов

передвижения<sup>1</sup>. Практическая возможность пользоваться услугами “мыслящих машин” предоставляется человеку в течение всего лишь 50 лет, а этого еще недостаточно для того, чтобы толкование слова “мыслить” распространилось и на машины.

Алан Тьюринг в своей знаменитой статье “*Computing Machinery and Intelligence*” [1520] указал, что вместо поиска ответа на вопрос, могут ли машины мыслить, мы должны интересоваться тем, могут ли машины пройти поведенческий тест интеллектуальности, который в дальнейшем получил название *теста Тьюринга*. Этот тест заключается в том, что программа в течение 5 минут участвует в разговоре (складывающемся из сообщений, которые передаются в оперативном режиме) с некоторым собеседником. Затем этот собеседник должен определить, проводился ли этот разговор с программой или с другим человеком; программа успешно проходит тест, если ей удается обмануть собеседника в 30% случаев. Тьюринг предсказал, что к 2000 году компьютер с объемом памяти в  $10^9$  единиц удастся запрограммировать настолько успешно, чтобы он прошел этот тест, но оказался неправ. Некоторых людей еще раньше удавалось водить за нос в течение 5 минут; например, программа Eliza и чатбот (робот-собеседник) Mgongz, действующий в Internet, не раз обманывали людей, которые так и не могли понять, что они общаются с программой, а программа Alice смогла даже одурачить судью на соревнованиях за приз Лёбнера (Loebner Prize) 2001 года. Но ни одной программе не удалось приблизиться к 30%-ному критерию в борьбе против специально обученных судей, да и в самой области искусственного интеллекта тесту Тьюринга уделяют все меньше внимания.

Тьюринг также исследовал широкий перечень вероятных возражений против самой возможности создания интеллектуальных машин, в том числе сумел предвидеть практически все возражения, которые были выдвинуты в течение полу века, прошедшего со времени появления его статьи. В этой главе будут рассмотрены некоторые из них.

### Довод, исходящий из неспособности

В “доводе, исходящем из неспособности”, выдвигается претензия, что “машина никогда не сможет выполнить действие X”. В качестве примеров действий X Тьюринг приводит следующий список.

Быть добрым, щедрым, красивым, дружелюбным, инициативным, иметь чувство юмора, отличать правду от лжи, совершать ошибки, влюбляться, наслаждаться землянкой и мороженым, заставлять влюбляться в себя, учиться на опыте, правильно употреблять слова, быть предметом собственных мыслей, проявлять такое же разнообразие в поведении, как и человек, создавать действительно что-то новое.

Тьюрингу пришлось воспользоваться своей интуицией для выдвижения предложений о том, что будет возможно в будущем, а мы имеем счастливую возможность оглянуться назад, чтобы узнать, чего уже удалось добиться компьютерам. Нельзя отрицать, что компьютеры в наши дни выполняют многие действия, которые раньше были прерогативой одних лишь людей. Программы играют в шахматы, шашки и другие игры, контролируют детали на сборочных конвейерах, проверяют правописание в документах, введенных с помощью текстового редактора, водят автомобили

<sup>1</sup> В русском языке эквивалент английского слова “swim” не носит такого ограничительного оттенка, поэтому может применяться и к судам.

и вертолеты, диагностируют заболевания, а также выполняют сотни других работ столь же хорошо, как люди, или даже лучше. Компьютеры сделали небольшие, но важные открытия в астрономии, математике, химии, минералогии, биологии, компьютерных науках и других областях. В каждом из этих случаев требовалось обладать способностями на уровне человека-эксперта.

С учетом того что мы знаем о компьютерах, не удивительно, что они легко справляются с такими комбинаторными задачами, как игра в шахматы. Но алгоритмы действуют также на уровнях не ниже человека при решении таких задач, которые, на первый взгляд, не позволяют обойтись без человеческого суждения или, как выразил это Тьюринг, заставляют “учиться на опыте” и требуют наличия способностей “отличать правду от лжи”. Еще в 1955 году Пауль Мель [1032] (см. также [600]) изучал процессы принятия решений обученными экспертами как субъективные задачи, аналогичные прогнозированию успешного овладения студентом программы обучения или повторного совершения преступления рецидивистом. В 19 из 20 рассмотренных им исследований Мель обнаружил, что простые алгоритмы статистического обучения (такие как алгоритмы линейной регрессии или наивные байесовские алгоритмы) вырабатывают лучшие предсказания, чем люди-эксперты. В американской Службе образовательного тестирования (Educational Testing Service) с 1999 года использовалась автоматизированная программа для выставления оценок по обзорным вопросам на экзаменах GMAT. Результаты работы программы согласовывались с результатами работы экзаменаторов, которым было поручено выставлять такие оценки, в 97% случаев, а этот уровень соответствует совпадению оценок двух экзаменаторов [212].

Очевидно, что компьютеры могут выполнять многие действия столь же успешно или даже лучше, чем люди, включая такие работы, в которых, по мнению людей, требуется огромная человеческая прозорливость и понимание. Это, безусловно, не означает, что компьютеры при выполнении этих работ проявляют прозорливость и понимание (указанные свойства не входят в состав поведения, о чем будет сказано ниже), но суть заключается в том, что первые предположения о содержании мыслительных процессов, требуемых для выработки данного конкретного поведения, часто оказываются ложными. Безусловно, верно также то, что во многих областях компьютеры все еще не добились значительного успеха (это еще мягко сказано), включая поставленную Тьюрингом задачу ведения разговора на свободную тему.

### Возражения, основанные на принципах математики

Благодаря работам Тьюринга [1518] и Гёделя [566] широко известно, что на некоторые математические вопросы нельзя даже в принципе найти ответ в рамках конкретных формальных систем. При этом наибольшую известность получила теорема Гёделя о неполноте (см. раздел 9.5). Кратко эту теорему можно сформулировать таким образом: для любой формальной аксиоматической системы  $F$ , достаточно мощной для того, чтобы в ней можно было представить арифметику, возможно сконструировать так называемое “предложение Гёделя”  $G(F)$  с описанными ниже свойствами.

- $G(F)$  — это предложение системы  $F$ , но оно не может быть доказано в рамках  $F$ .
- Если система  $F$  является непротиворечивой, то предложение  $G(F)$  — истинно.

Философы, в том числе Дж. Р. Лукас [960], утверждали, что эта теорема показывает, будто машины как мыслящие субъекты всегда будут стоять ниже людей, по-

скольку машины — это формальные системы, ограниченные в силу теоремы о неполноте (они не способны установить истинность относящегося к ним самим предложения Гёделя), а люди не имеют такого ограничения. Споры вокруг данного утверждения продолжались несколько десятилетий и породили огромное количество литературы, в том числе две книги математика сэра Роджера Пенроуза [1204], [1205], повторившего это утверждение с некоторыми новыми выкрутасами (такими как гипотеза, согласно которой человек отличается от машины, поскольку его мозг действует на основе квантовой гравитации). В этой главе мы рассмотрим только три из основных проблем, связанных с этим утверждением.

Во-первых, теорема Гёделя о неполноте распространяется только на формальные системы, достаточно мощные для того, чтобы в них можно было представить арифметику. К таким формальным системам относятся машины Тьюринга, поэтому утверждение Лукаса частично основано на том предположении, что компьютеры представляют собой машины Тьюринга. Это — хорошая аппроксимация, но не совсем оправданная. Машины Тьюринга являются бесконечными, а компьютеры конечны, поэтому любой компьютер может быть описан как (очень большая) система в пропозициональной логике, на которую не распространяется теорема Гёделя о неполноте.

Во-вторых, агенту не следует стыдиться того, что он не может определить истинность некоторого высказывания, тогда как другие агенты могут. Рассмотрим приведенное ниже предложение.

Дж.Р. Лукас не может неоспоримо утверждать, что это предложение истинно.

Если бы Лукас подтвердил истинность этого предложения, то он бы противоречил самому себе, поэтому Лукас не может неоспоримо подтвердить истинность данного предложения, а это означает, что оно должно быть истинным. (Это предложение не может быть ложным, поскольку, если бы Лукас не мог неоспоримо его подтвердить, то оно было бы истинным.) Таким образом, мы продемонстрировали, что есть такое предложение, которое Лукас не может неоспоримо подтвердить, тогда как другие люди (и машины) могут. Но из-за этого никто не вправе изменить своего мнения о Лукасе в худшую сторону. В качестве еще одного примера укажем, что ни один человек за всю свою жизнь не сможет вычислить сумму 10 миллиардов десятизначных чисел, а компьютер способен выполнить такую операцию за секунды. Тем не менее мы не рассматриваем этот факт как свидетельство фундаментального ограничения способности человека мыслить. Люди вели себя интеллектуально за тысячи лет до того, как изобрели математику, поэтому маловероятно, что способность формировать математические рассуждения играет более чем периферийную роль в том, что подразумевается под понятием интеллектуальности.

В-третьих (и это — наиболее важное возражение), даже если принять предположение, что компьютеры ограничены в том, что они способны доказать, нет никаких оснований считать, будто эти ограничения не распространяются на людей. Слишком легко вести этот спор, строго доказав, что формальная система не может выполнить действие X, а затем сообщив, что люди могут выполнить действие X, используя свои человеческие неформальные методы, но не дав ни одного свидетельства в пользу этого утверждения. И действительно, невозможно доказать, что на формальные рассуждения, проводимые людьми, не распространяется теорема Гёделя о неполноте, поскольку любое строгое доказательство само должно содержать формализацию способностей человеческого гения, которые, как многие утверждают, являются не-

формализуемым, и поэтому должно опровергать само себя. Таким образом, нам остается лишь прибегать к интуитивному представлению о том, что люди иногда способны проявлять сверхчеловеческие черты математического прозрения. Подобные утверждения выражаются в виде доводов: “мы должны быть уверены в том, что мы мыслим правильно, поскольку иначе мышление вообще становится невозможным” [961]. Но если об этом зашла речь, то давно известна склонность людей совершать ошибки. Это, безусловно, касается повседневной мыслительной деятельности, но справедливо также в отношении плодов математических рассуждений, полученных в результате упорной работы. Одним из известных примеров является теорема о раскраске карты четырьмя цветами. Математик Альфред Кемпэ опубликовал в 1879 году доказательство этой теоремы, которое было широко признано и стало одним из поводов к избранию этого математика в состав членов Королевского общества. Но в 1890 году Перси Хивуд указал на ошибку в этом доказательстве, и теорема оставалась недоказанной до 1977 года.

### Довод, исходящий из неформализуемости

Одно из наиболее важных и трудно оспоримых критических замечаний в адрес искусственного интеллекта как сферы приложения человеческих усилий было сформулировано Тьюрингом как довод, основанный на “неформализуемости поведения”. По сути это критическое замечание сводится к утверждению, что человеческое поведение является слишком сложным для того, чтобы его можно было описать с помощью какого-либо простого набора правил, а поскольку компьютеры не способны ни на что, кроме выполнения множества правил, они не способны и проявлять такое же интеллектуальное поведение, как люди. В искусственном интеллекте неспособность выразить все, что потребуется, в виде множества логических правил называют **проблемой спецификации** (см. главу 10).

Основными сторонниками этих взглядов были философы Хьюберт Дрейфус, который написал ряд влиятельных критических статей против искусственного интеллекта, в том числе *“What Computers Can’t Do”* (Что не способны делать компьютеры) [414] и *“What Computers Still Can’t Do”* [415], и его брат Стюарт, совместно с которым Хьюберт написал статью *“Mind Over Machine”* [416].

Положение дел, которое критиковали эти ученые, получило известность как “добрый старый искусственный интеллект”, или сокращенно GOFAI (Good Old-Fashioned AI); этот термин был предложен Хоглендом [631]. При этом предполагается, что в основе GOFAI лежит утверждение, будто все интеллектуальное поведение может быть представлено с помощью системы, которая формирует логические рассуждения на основании множества фактов и правил, описывающих рассматриваемую проблемную область. Поэтому GOFAI соответствует простейшему логическому агенту, описанному в главе 7. Дрейфус был прав, утверждая, что логические агенты действительно имеют слабое место, поскольку не позволяют решить проблему спецификации. Но как было показано в главе 13, для использования в открытых проблемных областях в большей степени подходят вероятностные системы формирования рассуждений. Поэтому критические замечания Дрейфуса относятся не к компьютерам как к таковым, а скорее к одному конкретному способу их программирования. Однако, вполне можно предположить, что более правильное название для статьи Дрейфуса, *“What First-Order Logical Rule-Based Systems Without Learning Can’t Do”* (Что не способны делать

системы на основе правил логики первого порядка, в которых не применяются средства обучения), было бы гораздо менее впечатляющим.

Согласно взглядам Дрейфуса, человеческий опыт подразумевает наличие знаний о некоторых правилах, но лишь в качестве “целостного контекста” (или “основы”), в рамках которого действуют люди. Он приводит пример корректного социального поведения при вручении и получении подарков: “Обычно люди, вручая подходящий к случаю подарок, действуют в рамках сложившихся в данном случае обстоятельств”. Очевидно, что люди обладают “непосредственным пониманием того, как следует действовать и чего следует ожидать”. Такое же утверждение он выдвигает в контексте игры в шахматы: “Шахматисту среднего уровня может потребоваться обдумать следующий ход, а гроссмейстер просто видит, что положение на доске само требует определенного хода …правильный ответ сам складывается в его голове”. Безусловно, нельзя отрицать, что основная часть мыслительных процессов лица, готовящего подарок, или гроссмейстера, выбирающего ход, осуществляется на уровне, недоступном для самоанализа со стороны пытливого разума. Но из этого не следует, что сами эти мыслительные процессы не происходят. Важный вопрос, на который не отвечает Дрейфус, состоит в том, как правильный ход появляется в голове гроссмейстера. Напомним читателю один из комментариев Дэниела Деннета [389], приведенный ниже.

Создается впечатление, будто философы взяли на себя роль толкователей приемов фокусников. Когда их спрашивают, как фокусник ставит свой трюк с распилюванием ассистентки пополам, они объясняют, что в этом нет ничего сложного: фокусник фактически никого не распиливает; он просто заставляет людей верить, что он это делает. Если же философов спрашивают: “Но как ему удается создать такое впечатление?”, они отвечают: “Мы в этом не компетентны”.

В статье братьев Дрейфус [416] предложен пятиэтапный процесс приобретения опыта, который начинается с обработки полученной информации на основе правил (осуществляемой по такому же принципу, как в GOFAI) и заканчивается приобретением способности мгновенно выбирать правильные ответы. Но внося свое предложение, братья Дрейфус по сути перешли из разряда критиков искусственного интеллекта в разряд его теоретиков — они предложили архитектуру нейронной сети, организованную в виде огромной “библиотеки примеров”, указав при этом на несколько проблем. К счастью, все указанные ими проблемы полностью решены, причем некоторые частично, а другие полностью. Упомянутые братьями Дрейфус проблемы перечислены ниже.

1. Качественное обобщение на основании примеров не может быть достигнуто без фоновых знаний. Дрейфусы утверждают, что не существует способа, позволяющего использовать фоновые знания в процессе обучения нейронной сети. А в действительности, как было описано в главе 19, уже разработаны такие методы, которые позволяют использовать априорные знания в алгоритмах обучения. Тем не менее эти методы основаны на том, что в наличии имеются знания, представленные в явной форме, а этот подход братья Дрейфус упорно отрицают. По мнению авторов настоящей книги, взгляды этих ученых являются основательной причиной для серьезного перепроектирования современных моделей нейронной обработки информации, для того чтобы в них можно было воспользоваться знаниями, полученными ранее в процессе обу-

чения, по такому же принципу, как такие знания используются в других алгоритмах обучения.

2. Обучение нейронной сети представляет собой одну из разновидностей контролируемого обучения (см. главу 18), для которой требуется заблаговременное выявление релевантных входных данных и правильных выходных данных. Поэтому, по утверждению братьев Дрейфус, система обучения нейронной сети не может действовать автономно, без помощи учителя-человека. В действительности обучение без учителя может осуществляться с помощью методов **неконтролируемого обучения** (см. главу 20) и **обучения с подкреплением** (см. главу 21).
3. Производительность алгоритмов обучения снижается при использовании большого количества характеристик, а если выбрано лишь подмножество характеристик, то, по словам этих ученых, “не существует способа введения новых характеристик в том случае, если будет обнаружено, что текущее множество не позволяет учитывать некоторые факты, усваиваемые в процессе обучения”. В действительности с большими множествами характеристик очень успешноправляются такие новые методы, как машины поддерживающих векторов. Как было показано в главе 19, существует также принципиальная возможность вырабатывать новые характеристики, хотя для этого требуется гораздо больше усилий.
4. Мозг способен направлять свои сенсоры на поиск релевантной информации и обрабатывать ее для извлечения аспектов, релевантных для текущей ситуации. Но Дрейфусы утверждают, что “в настоящее время неизвестны детали этого механизма и нет даже таких гипотез о его работе, которые направили бы исследования искусственного интеллекта по правильному пути”. В действительности проблеме выбора правильной ориентации сенсоров посвящена область активного зрения, основанная на теории стоимости информации (см. главу 16), а полученные теоретические результаты уже были применены при создании некоторых роботов.

В конечном итоге многие проблемы, на которых сосредоточились братья Дрейфус (фоновые обыденные знания, проблема спецификации, неопределенность, обучение, компилированные формы средств принятия решений, важность применения агентов, реагирующих на текущую ситуацию, а не беспилотных машин логического вывода), уже были решены, а достигнутые результаты воплощены в стандартных проектах интеллектуальных агентов. По мнению авторов настоящей книги, это — свидетельство прогресса искусственного интеллекта, а не подтверждение non-существенности поставленной перед ним цели.

## 26.2. СИЛЬНЫЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ: МОГУТ ЛИ МАШИНЫ ПО-НАСТОЯЩЕМУ МЫСЛИТЬ?

Многие философы утверждали, что даже машина, которая пройдет тест Тьюринга, все равно фактически будет не мыслить, а лишь имитировать мышление. Тьюринг предвидел и это возражение против искусственного интеллекта. В частно-

сти, он процитировал приведенный ниже фрагмент речи профессора Джейфри Джейферсона [726].

Мы сможем согласиться, что машина равна мозгу, лишь после того, как она будет в состоянии написать сонет или сочинить концерт под воздействием своих мыслей и эмоций, а не благодаря случайному совпадению нужных символов; под этим подразумевается, что машина должна не только написать подобное произведение, но и понимать, что оно ею написано.

Тьюринг назвал это возражение доводом, основанным на понятии **сознания**; согласно этому доводу, машина должна понимать свои собственные психические состояния и действия. Безусловно, сознание — это важная тема, но ключевая идея Джейфера сона фактически касается проблемы **феноменологии**, или изучения непосредственного опыта, т.е. этот ученый требует, чтобы машина действительно ощущала эмоции. Другие ученые сосредоточиваются на проблеме **целенаправленности**, т.е. на вопросе о том, действительно ли приписываемые машине убеждения, желания и другие внутренние представления касаются “чего-то”, существующего в реальном мире.

Ответ Тьюринга на это возражение весьма интересен. Он мог продемонстрировать причины, по которым машины на самом деле способны были бы действовать сознательно (либо с точки зрения феноменологии, либо с точки зрения целенаправленности). Вместо этого он указал, что данный вопрос столь же некорректен, как и вопрос о том, могут ли машины мыслить. К тому же, на каком основании мы требуем применения к машинам более высоких стандартов, чем к людям? В конечном итоге в повседневной жизни мы никогда не получаем каких-либо прямых свидетельств о внутреннем психическом состоянии других людей. Тем не менее Тьюринг заявил: “Вместо ведения бесконечных споров на эту тему обычно принято заключать **дженрльменское соглашение** и считать, что мыслят все”.

Тьюринг утверждает, что Джейферсон согласился бы распространить это джентльменское соглашение на машины, только если бы имел опыт общения с теми из них, которые действуют интеллектуально. Он процитировал действительно происходивший приведенный ниже диалог человека с машиной, который считается такой неотъемлемой частью передающихся из уст в уста легенд искусственного интеллекта, что мы просто обязаны его включить в эту главу.

Человек. In the first line of your sonnet which reads “shall I compare thee to a summer's day”, would not a “spring day” do as well or better? (В первой строке вашего сонета сказано “я хочу сравнить вас с летним днем”; не было бы так же хорошо или даже лучше сказать “с весенним днем”?)

Машина. It wouldn't scan. (Нарушилась бы ритмика.)

Человек. How about “a winter's day”. That would scan all right. (А как насчет слов “с зимним днем”. Ритмика бы не нарушилась.)

Машина. Yes, but nobody wants to be compared to a winter's day. (Да, но никто не хочет, чтобы его сравнивали с зимним днем.)

Человек. Would you say Mr. Pickwick reminded you of Christmas? (Вы хотите сказать, что мистер Пиквик напомнил вам о Рождестве?)

Машина. In a way. (В определенном смысле.)

Человек. Yet Christmas is a winter's day, and I do not think Mr. Pickwick would mind the comparison. (Но все же Рождество — зимний день, и я не думаю, что мистер Пиквик возражал бы против такого сравнения.)

Машина. I don't think you're serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas. (Я не думаю, что вы говорите серьезно. Под зимним днем подразумевается обычный зимний день, а не такой особый день, как Рождество.)

В заключение Тьюринг отметил, что вопрос о сознании является трудноразрешимым, но опроверг мнение о том, что он имеет большую значимость для практики искусственного интеллекта: “Я отнюдь не желаю, чтобы мои слова были истолкованы таким образом, будто я не считаю проблему сознания сложной загадкой... но я не думаю, что нужно обязательно разгадать все подобные загадки, прежде чем мы сможем ответить на тот вопрос, о котором идет речь в данной статье”. Авторы настоящей книги согласны с Тьюрингом: мы заинтересованы в создании программ, которые действуют интеллектуально, а не в том, чтобы дать кому-то повод считать эти действия настоящими или имитированными. С другой стороны, эта проблема остается предметом острого интереса для многих философов. Для того чтобы понять суть такой заинтересованности, рассмотрим вопрос о том, считаются ли реальными другие искусственно созданные объекты.

В 1848 году Фредерик Вёлер впервые синтезировал искусственную мочевину. Это достижение было очень важным, поскольку стало доказательством единства органической и неорганической химии, а также позволило поставить точку в вопросе, который до сих пор был предметом горячих дебатов. После успешного осуществления этого синтеза химики согласились, что искусственная мочевина действительно представляет собой мочевину, поскольку обладает всеми правильными физическими свойствами. Аналогичным образом, нельзя отрицать, что искусственные подслащающие вещества действительно являются подслащающими веществами, а искусственное оплодотворение (еще один термин с аббревиатурой AI — Artificial Insemination) действительно является оплодотворением. С другой стороны, искусственные цветы — это не цветы, и, как указал Дэниел Деннет (Daniel Dennett), искусственное вино Шато Латур — это не вино Шато Латур, даже если образцы того и другого нельзя отличить друг от друга с помощью химического анализа, просто потому, что оно не было изготовлено в должном месте правильным способом. А искусственно выполненный рисунок Пикассо — это не рисунок Пикассо, независимо от того, похож он на оригинал или нет.

На основании изложенного можно сделать вывод, что в некоторых случаях важно лишь поведение искусственного объекта, а в других случаях играет роль также происхождение искусственного объекта. То, в каком случае приобретает важность последний фактор, по-видимому, обусловлено лишь принятыми соглашениями. А когда речь идет об искусственном разуме, мы не можем опереться на принятые по этому поводу соглашения, и нам остается полагаться лишь на интуитивные предположения. Философ Джон Сирл [1376, с. 37, 38] выдвинул следующее весьма убедительное предположение.

Никто не думает, что компьютерная имитация грозы заставит его вымокнуть... так почему же люди, будучи в здравом уме, могут предположить, что компьютерная имитация мыслительных процессов действительно представляет собой мыслительные процессы?

Безусловно, нельзя не согласиться, что компьютерные имитации гроз не заставят нас вымокнуть, но не совсем понятно, как можно перенести эту аналогию на компьютерные имитации мыслительных процессов. К тому же создаваемые в Голливуде имитации гроз, в которых используются опрыскиватели и воздуходувки, действительно заставляют актеров вымокнуть. Большинство людей, не задумываясь, скажет,

что компьютерная имитация сложения является сложением, а компьютерная имитация шахматной игры является шахматной игрой. Что больше напоминают мыслительные процессы — грозы или абстрактные операции, такие как арифметическое сложение и игра в шахматы? С чем их следует сравнивать — со штучными изделиями, такими как вино Шато Латур и картины Пикассо, или с массовой продукцией, такой как мочевина? Ответы на все эти вопросы зависят от принятой теории психических состояний и процессов.

В теории **функционализма** утверждается, что психическим состоянием является любое промежуточное причинное условие, связывающее входные и выходные данные. Согласно теории функционализма, любые две системы с изоморфными причинными процессами должны иметь одни и те же психические состояния. Поэтому компьютерная программа может иметь такие же психические состояния, как и человек. Безусловно, мы еще не дали определения того, что действительно подразумевается под термином “изоморфный”, но основное допущение состоит в том, что существует некоторый уровень абстракции, ниже которого конкретные детали реализации не имеют значения; при условии, что ниже этого уровня процессы являются изоморфными, возникают одни и те же психические состояния.

В отличие от этого, в теории **биологического натурализма** утверждается, что психические состояния представляют собой высокоуровневые эмерджентные характеристики, которые вызваны неврологическими процессами низкого уровня в нейронах, и ведущую роль в этих процессах играют некоторые (неопределенные) свойства нейронов. Это означает, что психические состояния не могут быть продублированы лишь на основе некоторой программы, имеющей такую же функциональную структуру и проявляющей такое же поведение, выраженное в виде входных/выходных данных; мы должны потребовать, чтобы эта программа эксплуатировалась в архитектуре, обладающей такой же причинной мощью, как и нейроны. В указанной теории ничего не говорится о том, почему нейроны обладают этой причинной мощью, а также о том, существуют ли другие физические воплощения, которые могут иметь или не иметь эту причинную мощь.

Чтобы проанализировать эти две точки зрения, вначале рассмотрим одну из самых старых проблем в области философии разума, а затем обратимся к трем мысленным экспериментам.

## Проблема разума и тела

**Проблема разума и тела** касается вопроса о том, как психические состояния и процессы связаны с физическими состояниями и процессами (а именно с процессами, происходящими в мозгу). Игнорируя достаточно высокую сложность этой проблемы, обобщим ее до уровня проблемы “архитектуры разума”, что позволит нам вести речь о том, могут ли машины иметь разум.

Почему проблема разума и тела является такой сложной? Первую сложность обнаружил еще Рене Декарт, который размышлял над тем, как бессмертная душа взаимодействует со смертным телом, и пришел к выводу, что душа и тело — это два различных типа субстанций; в этом состоит так называемая теория **дуализма**. С другой стороны, теория **монизма**, часто называемая **материализмом**, основана на том, что таких субстанций, как нематериальные души, просто не бывает; в мире имеются только материальные объекты. Поэтому все психические состояния

(возникающие, когда человек испытывает боль, осознает себя как скачущий на лошади или думает, что Вена — столица Австрии) представляют собой состояния мозга. Джон Сирл метко сформулировал эту идею в виде лозунга “Мозг рождает разум”.

Но материализму приходится сталкиваться с двумя серьезными препятствиями. Первым из них является проблема **свободной воли**: как так может оказаться, что чисто физический разум, каждое преобразование в котором строго управляетяется законами физики, все еще сохраняет какую-то свободу выбора? Большинство философов рассматривают эту проблему как требующую тщательного переопределения наших наивных представлений о свободной воле, а не представляющую собой какое-либо покушение на материализм. Вторым препятствием является проблема, касающаяся общего вопроса о **сознании** (а также связанных с ним, но не идентичных вопросов **понимания и самосознания**). В наиболее простой формулировке этот вопрос состоит в том, почему человек ощущает себя как имеющий определенные мыслительные состояния и вместе с тем не чувствует себя как имеющий какие-то другие физические состояния (например, не считает себя камнем).

Чтобы приступить к поиску ответа на подобные вопросы, мы должны найти способы вести речь о состояниях мозга на уровнях, более абстрактных, чем конкретные конфигурации всех атомов мозга определенного человека в конкретное время. Например, когда я размышляю о столице Австрии, мой мозг подвергается бесчисленному количеству мельчайших изменений за время, прошедшее от одной пикосекунды до другой, но это не приводит к качественному изменению состояния мозга. Для того чтобы учесть возможные ситуации, необходимо ввести понятие *типов состояния мозга*, в рамках которого можно было бы судить, принадлежат ли два состояния мозга к одному и тому же или к разным типам. Различные ученые имеют несовпадающие взгляды на то, что подразумевается в данном случае под типом. Но почти все ученые считают, что если взять живой мозг и заменить в нем некоторые из атомов углерода новым множеством атомов углерода<sup>2</sup>, то психическое состояние мозга не изменится. Эта гипотеза вполне оправдана, поскольку в действительности в мозгу непрерывно происходит замена атомов в результате метаболических процессов, тем не менее такой процесс, по-видимому, не вызывает существенных психических расстройств.

Теперь рассмотрим конкретный вид психического состояния: **пропозициональные позиции** (впервые описанные в главе 10), которые также известны как **ментальные состояния**. Таковыми являются состояния, подобные убежденности, уверенности, желанию, чувству страха и т.д., которые относятся к некоторому аспекту внешнего мира. Например, уверенность в том, что Вена — столица Австрии, — это убеждение, касающееся конкретного города и его статуса. Нас интересует вопрос, могут ли компьютеры иметь ментальные состояния, чтобы можно было понять, как охарактеризовать эти состояния. Например, можно утверждать, что психическое состояние, в котором я хочу гамбургер, отличается от состояния, в котором я хочу пиццу, поскольку гамбургер и пицца в реальном мире отличаются друг от друга. А если верно такое утверждение, то ментальные состояния имеют необходимую связь с относящимися к ним объектами во внешнем мире. С другой стороны, всего лишь за несколько абзацев перед этим было сформулировано утверждение, что психические состояния представляют собой состояния мозга, поэтому идентичность или неиден-

<sup>2</sup> Возможно, даже атомами другого изотопа углерода, как иногда бывает в экспериментах по сканированию мозга.

тичность психических состояний должна определяться полностью “внутри самой головы”, без ссылок на реальный мир. Чтобы лучше изучить эту дилемму, обратимся к мысленному эксперименту, позволяющему отделить целенаправленные состояния от относящихся к ним внешних объектов.

### Эксперимент “мозг в колбе”

Допустим, что при желании мозг человека можно отделить от тела сразу после рождения и поместить в колбу, искусно спроектированную для этой цели. В этой колбе мозг получает питание и опору, а она позволяет ему расти и развиваться. Наряду с тем в мозг подаются электронные сигналы от компьютера, моделирующего полностью вымышленный мир, а моторные сигналы от мозга перехватываются и используются для модификации этой имитации должным образом<sup>3</sup>. В таком случае мозг может иметь психическое состояние *DyingFor(Me, Hamburger)* (страстное желание получить гамбургер), даже несмотря на то, что у него нет тела, которое испытывало бы чувство голода, а также вкусовых рецепторов, чтобы ощутить вкус, к тому же в реальном для мозга (но фактически имитируемом) мире могло бы просто не оказаться гамбургера. Было бы это психическое состояние таким же, какое испытывает мозг в живом теле?

Один из способов разрешения этой дилеммы состоит в использовании гипотезы о том, что содержимое психических состояний можно интерпретировать с двух разных точек зрения. В подходе на основе **широкого анализа содержимого** психические состояния интерпретируются с точки зрения всезнающего внешнего наблюдателя, имеющего доступ к информации обо всей ситуации, который способен замечать любые различия в мире. Поэтому при широком анализе содержимого чувства мозга, живущего в колбе, можно отличить от чувств “обычного” человека. А при **узком анализе содержимого** учитывается только внутренняя субъективная точка зрения, и при таком подходе чувства, испытываемые тем и другим мозгом, остаются одинаковыми.

Уверенность в том, что гамбургер — желанная пища, имеет определенный характер связи с конкретным объектом, поскольку обязательно должен быть кто-то, обладающий уверенностью в таком свойстве гамбургера. Переходя к рассуждениям такого рода, мы вступаем в область познания **качества**, или собственного опыта (в англоязычной литературе для обозначения этого понятия применяется термин “*qualia*”, происходящий от латинского слова, которое переводится примерно как “именно такие”). Предположим, что в результате какого-то нарушения в нервных путях сетчатки и мозга неким лицом X воспринимается как красный тот цвет, который лицо Y воспринимает как зеленый, и наоборот. В таком случае, увидев один и тот же сигнал светофора, оба они действуют одинаково, но воспринимаемый ими опыт должен быть в определенной степени разным. Оба эти лица могут согласиться, что полученные ими данные восприятия говорят о том, что “свет на светофоре — красный”, но сами эти восприятия остаются разными. Поэтому неясно, свидетельствует ли это о том, что они имеют одинаковые или разные психические состояния.

<sup>3</sup> Описанная ситуация может быть знакома тем, кто смотрел фильм *The Matrix* (Матрица), вышедший на экраны в 1999 году.

Теперь перейдем еще к одному мысленному эксперименту, который относится к вопросу о том, могут ли иметь психические состояния физические объекты, отличные от нейронов человека.

### Эксперимент с протезом мозга

Мысленный эксперимент с протезом мозга был предложен в середине 1970-х Кларком Глаймором и описан в трудах Джона Сирла [1376], но его чаще всего связывают с работой Ханса Моравека [1084]. Этот эксперимент заключается в следующем: предположим, что нейрофизиология достигла огромного уровня развития, на котором существует идеальное понимание взаимодействия входных и выходных сигналов и связей между всеми нейронами в мозгу человека. Предположим также, что существует возможность создавать микроскопические электронные устройства, имитирующие поведение нейрона, которые можно незаметно для человека подключать к его нервной ткани. Наконец, предположим, что некая чудесная хирургическая техника позволяет заменять отдельные нейроны соответствующими электронными устройствами, не нарушая работу мозга в целом. Эксперимент состоит в постепенной замене всех нейронов в голове человека электронными устройствами, а затем в обратном выполнении этого процесса для возврата испытуемого субъекта в его нормальное биологическое состояние.

Нас интересует как внешнее поведение, так и внутренний опыт этого субъекта во время операции и после нее. Согласно определению этого эксперимента, внешнее поведение субъекта должно оставаться неизменным по сравнению с тем, которое наблюдалось бы в том случае, если бы эта операция не выполнялась<sup>4</sup>. Итак, несмотря на то, что в присутствии или отсутствии сознания не так уж легко убедиться, будучи сторонним наблюдателем, сам субъект эксперимента должен по крайней мере иметь способность зарегистрировать любые изменения в своем восприятии собственного сознания. Очевидно, что возникает прямой конфликт интуитивных представлений о том, что может произойти. Моравек, будучи специалистом в области робототехники и приверженцем взглядов функционалистов, убежден в том, что сознание субъекта, подвергающегося эксперименту, останется незатронутым, а Сирл, философ и натуралист-биолог, столь же твердо убежден, что сознание у субъекта эксперимента постепенно исчезнет, о чем свидетельствует приведенная ниже цитата из его работы.

Вы обнаружите, к своему полному изумлению, что действительно теряете контроль над своим внешним поведением. Например, вы заметите, что при проверке врачами вашего зрения вам, допустим, скажут: “Мы держим перед вами объект красного цвета; пожалуйста, сообщите нам, что вы видите”. Вы захотите крикнуть: “Я ничего не вижу. Я полностью ослеп”, но услышите, как ваш голос говорит, полностью не подчиняясь вашему контролю: “Я вижу перед собой объект красного цвета...” Ваше сознание постепенно сужается и исчезает, тогда как внешне наблюдаемое поведение остается неизменным [1379].

Но существует возможность вести этот спор, опираясь не только на интуицию. Во-первых, следует отметить, что внешнее поведение будет оставаться одинаковым в процессе того, как субъект постепенно теряет сознание, только в том случае, если

<sup>4</sup> Можно также представить себе, что в эксперименте участвует идентичный “контрольный” субъект, над которым якобы выполняется операция, но фактически она не проводится. Это позволяет сравнивать поведение двух субъектов.

воля субъекта исчезает мгновенно и полностью; в противном случае сужение сознания должно было бы отразиться на внешнем поведении; это означает, что испытуемый должен был бы закричать: “Помогите, я теряю сознание!” или нечто в этом роде. Такая гипотеза мгновенного исчезновения воли в результате постепенной замены одного за другим отдельных нейронов кажется маловероятной.

Во-вторых, рассмотрим, что произойдет, если мы будем задавать субъекту вопросы, касающиеся того, как он сам ощущает наличие у него сознания в тот период, когда у него не останется ни одного настоящего нейрона. Согласно условиям эксперимента, мы должны получать примерно такие ответы: “Я чувствую себя прекрасно. Я должен также отметить, что немного удивлен, поскольку верил в истинность доводов Сирла”. Еще один вариант состоит в том, что мы могли бы уколоть субъекта заостренной палочкой и услышать ответ: “Ой, больно”. Теперь, если вернуться к обычной жизни, то скептик может возразить, что подобные выходные данные могут быть получены и от программ искусственного интеллекта как простые результаты принятых соглашений. Действительно, совсем не сложно предусмотреть, например, такое правило: “Если на датчике номер 12 появится сигнал с высокой интенсивностью, то выдать выходные данные «Ой, больно»”, но весь смысл рассматриваемого эксперимента состоит в том, что мы продублировали функциональные свойства обычного человеческого мозга, поэтому предполагается, что электронный мозг не содержит таких структур, в которых реализованы подобные соглашения. Это означает, что нужно как-то объяснить, чем обусловлены проявления сознания, вырабатываемые электронным мозгом, которые основаны лишь на функциональных свойствах нейронов. И это объяснение должно также распространяться на настоящий мозг, который имеет такие же функциональные свойства. На наш взгляд, можно сделать только два приведенных ниже возможных вывода.

1. Причинные механизмы формирования сознания, которые вырабатывают выходные данные такого рода в обычном мозгу, все еще продолжают действовать в электронной версии мозга, поэтому последняя также обладает сознанием.
2. Осознаваемые психические события в обычном мозгу не имеют причинной связи с поведением, а поскольку они отсутствуют также в электронном мозгу, последний не обладает сознанием.

Хотя нельзя исключить вторую возможность, при данном подходе сознание сводится к тому, что философы называют ~~э~~ эпифеноменальной (выраженной в качестве побочного явления) ролью — как будто что-то происходит, но не отбрасывает тени, которая должна была бы появиться в наблюдаемом мире. Кроме того, если сознание действительно эпифеноменально, то в мозгу должен находиться второй, бессознательный механизм, с помощью которого и формируется восклицание “Ой, больно”, когда человека колют заостренной палочкой.

В-третьих, рассмотрим ситуацию, возникшую после того, как операция проделана в обратном направлении и субъект снова имеет обычный мозг. И в этом случае внешнее поведение субъекта должно быть, по определению, таким же, как если бы операция вовсе не проводилась. В частности, мы были бы вправе задать субъекту такие вопросы: “Что вы чувствовали во время операции? Помните, как вас укололи заостренной палочкой?” Субъект должен точно помнить фактический характер своего осознанного опыта, включая качественную сторону этого опыта, несмотря на тот факт, что, согласно Сирлу, такого опыта не должно быть.

Сирл мог бы возразить, что мы не определили эксперимент должным образом. Если бы настоящие нейроны, скажем, прекращали действовать в том промежутке времени, когда они были извлечены, а затем снова помещены в мозг, то, безусловно, они не могли бы “запомнить” опыт, полученный в это время. Чтобы учесть это обстоятельство, необходимо обеспечить обновление состояния нейронов в соответствии с изменением внутреннего состояния искусственных нейронов, которыми они заменялись. Если бы в таком случае предполагалось наличие “нефункциональных” аспектов реальных нейронов, которые привели бы к поведению, функционально отличному от того, которое наблюдалось, пока искусственные нейроны были бы еще на месте, то налицо было бы простое приведение к абсурду, поскольку означало бы, что искусственные нейроны функционально не эквивалентны настоящим нейронам (одно из возможных возражений против этого довода приведено в упр. 26.3).

Патрисия Чарчленд [260] указала, что приведенные выше доводы, основанные на теории функционализма и применяемые на уровне нейронов, могут также использоваться на уровне любой более крупной функциональной единицы — группы нейронов, раздела мозга, доли, полушария или целого мозга. Это означает, что, соглашившись с доводом, что эксперимент с протезом мозга демонстрирует наличие сознания у мозга, в котором нейроны заменены электронными компонентами, мы должны также согласиться, что сознание сохраняется, если весь мозг заменяется схемой, в которой входные данные отображаются на выходные с помощью огромной поисковой таблицы. Такое представление неприемлемо для многих людей (включая самого Тьюринга), интуиция которых подсказывает, что поисковые таблицы вряд ли могут иметь сознание или, по меньшей мере, что сознательный опыт, сформированный во время поиска в таблице, не сопоставим с опытом, сформированным в процессе работы системы, которая может быть описана (даже в примитивном, вычислительном смысле) как манипулирование с убеждениями, результатами самоанализа, целями и тому подобными явлениями, которые формируются мозгом. Эти замечания свидетельствуют о том, что эксперимент с протезом мозга может быть эффективным средством, подкрепляющим нашу интуицию, только если в нем не предусматривается замена сразу всего мозга, но это и не означает, что в данном эксперименте может лишь рассматриваться замена одних атомов другими, как хочет нас заставить считать Сирл.

## Китайская комната

Последний мысленный эксперимент, который будет описан в данной главе, по-видимому, является самым известным из всех. Идея этого эксперимента принадлежит Джону Сирлу [1376], описавшему гипотетическую систему, в отношении которой любому наблюдателю ясно, что она работает под управлением какой-то программы и успешно проходит тест Тьюринга, но также ясно (согласно Сирлу), что эта система не понимает смысла каких-либо из ее входных или выходных данных. На основании этого Сирл делает вывод, что работа системы под управлением приемлемой программы (т.е. программы, вырабатывающей правильные выходные данные) не является достаточным условием для обладания разумом.

Система состоит из человека, который понимает только английский язык, снабжен книгой с правилами, написанной на английском языке, а в его распоряжении находятся разные стопки бумаг, причем некоторые из них пусты, а другие заполнены

ны описаниями, не поддающимися расшифровке. (Таким образом, человек играет роль процессора компьютера, книга правил представляет собой программу, а стопки бумаг соответствуют запоминающему устройству.) Система находится в комнате с небольшим отверстием, выходящим наружу. Через отверстие появляются полоски бумаги с символами, не поддающимися расшифровке. Человек находит совпадающие с ними символы в книге правил и следует обнаруженным в ней инструкциям. Инструкции могут предусматривать задания по написанию символов на новых полосках бумаги, поиску символов в стопках, переупорядочению стопок и т.д. В конечном итоге инструкции диктуют необходимость написания одного или нескольких символов на листе бумаги, который снова передается во внешний мир.

До сих пор все шло нормально. Но из внешнего мира мы видим систему, которая принимает входные данные в форме предложений на китайском языке и формирует ответы на китайском, которые внешне кажутся “интеллектуальными”, как и беседа, мысленно представленная Тьюрингом<sup>5</sup>. После этого Сирл проводит следующие рассуждения: человек, сидящий в комнате, не понимает китайского (это дано). Книга правил и стопки бумаги, будучи просто листами бумаги, не понимают китайского. Поэтому речь не может идти о каком-либо понимании китайского языка. *Поэтому, согласно Сирлу, эксплуатация даже подходящей программы не обязательно приводит к развитию понимания.*

Как и Тьюринг, Сирл рассмотрел и попытался опровергнуть целый ряд ответов на его доводы. Некоторые комментаторы, включая Джона Маккарти и Роберта Виленского, выдвинули предложения, которые Сирл назвал *системными ответами*. Их возражение состоит в том, что человека, сидящего в комнате, безусловно, можно спросить, понимает ли он китайский язык, но это аналогично тому, как если бы процессор компьютера спросили, может ли он извлекать кубические корни. В обоих случаях ответ является отрицательным и в обоих случаях, согласно системному ответу, вся система обладает способностью, которая была предметом вопроса. Безусловно, если задать системе с китайской комнатой вопрос на китайском языке, знает ли она китайский, ответ будет утвердительным (и сформулированным на живом китайском языке). Согласно корректному соглашению Тьюринга, этого должно быть достаточно. Ответ Сирла — это возврат к той идее, что понимание не заключено в мозгу человека, сидящего в китайской комнате, и не может быть заключено в стопках бумаги, поэтому не может быть никакого понимания. Далее Сирл указывает, что можно представить себе ситуацию, когда человек запоминает книгу правил и содержимое всех стопок бумаги, поэтому больше нет ни одного объекта, которому можно было бы приписать понимание, кроме самого человека; но и после этого, если ему будет задан вопрос (на английском языке), понимает ли он китайский, ответ будет отрицательным.

Теперь перейдем к реальному анализу этой проблемы. Переход от эксперимента с использованием стопок бумаги к эксперименту с запоминанием — это просто попытка сбить читателя с толку, поскольку обе формы представляют собой варианты физического воплощения работающей программы. Фактические утверждения, выдвинутые Сирлом, опираются на четыре приведенных ниже аксиомы [1378].

---

<sup>5</sup> Тот факт, что стопки бумаги могут оказаться больше всей нашей планеты, а выработка ответов может занять миллионы лет, не имеет отношения к логической структуре данного эксперимента. Одной из целей обучения философии является выработка тщательно отточенного понимания того, какие выражения являются обоснованными и какие нет.

1. Компьютерные программы представляют собой формальные, синтаксические сущности.
2. Разум имеет мыслительное содержание, или семантику.
3. Синтаксиса как такового не достаточно для семантики.
4. Мозг порождает разум.

На основании первых трех аксиом Сирл делает вывод, что программы не могут служить достаточным условием для появления разума. Иными словами, агент, в котором функционирует какая-то программа, может оказаться разумным, но он не обязательно становится разумным лишь в силу того, что в нем работает программа. На основании четвертой аксиомы Сирл делает вывод: “Любая другая система, способная порождать разум, должна обладать причинной мощью (по меньшей мере), эквивалентной той, какой обладает мозг”. На основании этого он делает вывод, что любой искусственный мозг должен воплощать в себе такую же причинную мощь, как и мозг, а не только работать под управлением конкретной программы, и что мозг человека не вырабатывает мыслительные феномены исключительно благодаря тому, что в нем функционирует какая-то программа.

Вывод о том, что применения программ недостаточно для создания разума, действительно следует из этих аксиом, если допускается их вольная интерпретация. Но само обоснование вывода проведено неудовлетворительно — все, что смог доказать Сирл, состоит в том, что если явно отвергнуты принципы функционализма (как было сделано в его третьей аксиоме), то заключение, согласно которому объекты, отличные от мозга, порождают разум, становится необязательным. Это предположение вполне обосновано, поэтому вся дискуссия сводится к тому, может ли быть принята третья аксиома. Согласно Сирлу, весь смысл эксперимента с китайской комнатой состоит в предоставлении интуитивного обоснования для третьей аксиомы. Но реакция других исследователей показывает, что такие интуитивные представления близки по духу только тем, кто уже был склонен соглашаться с идеей, что программы, взятые в чистом виде, не способны вырабатывать истинное понимание.

Еще раз отметим, что цель эксперимента с китайской комнатой состоит в опровержении понятия сильного искусственного интеллекта — утверждения, что эксплуатация программы подходящего типа обязательно приводит к появлению разума. Этот мысленный эксперимент проводится путем демонстрации внешне интеллектуальной системы, в которой функционирует программа подходящего типа, но в отношении этой системы, согласно Сирлу, можно явно показать, что она не обладает разумом. Для этого Сирл прибегает к интуиции, а не к доказательству; он как будто говорит нам: “достаточно взглянуть на эту комнату; разве в ней может быть разум?” Но точно такой же довод можно привести и применительно к мозгу — достаточно взглянуть на этот конгломерат клеток (или атомов), работающих вслепую в соответствии с законами биохимии (или физики); разве в нем может быть разум? Почему в куске мозга может быть разум, а в куске печени — нет?

Более того, Сирл, соглашаясь с тем, что материалы, отличные от нейронов, могут в принципе быть носителем разума, ослабляет свои доводы еще больше, по двум причинам: во-первых, нам приходится руководствоваться только интуицией Сирла (или своей собственной), чтобы доказать, что в китайской комнате отсутствует разум, и, во-вторых, даже если мы решим, что в этой комнате нет разума, такой вывод

не позволит нам узнать что-либо о том, не будет ли программа, работающая в какой-то другой физической среде (включая компьютер), иметь разум.

Сирл допускает логическую возможность того, что мозг действительно реализует программу искусственного интеллекта традиционного типа, но та же программа, работающая в машине неподходящего типа, не создает разум. Сирл отрицает то, будто он верит, что “машины не могут иметь разума”, скорее он утверждает, что некоторые машины имеют разум, например люди — это биологические машины, обладающие разумом. Но он также оставляет нас в неведении относительно того, какого же типа машины подходят или не подходят под определение понятия разумных машин.

## 26.3. ЭТИЧЕСКИЕ И МОРАЛЬНЫЕ ПОСЛЕДСТВИЯ РАЗРАБОТКИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

---

До сих пор в этой главе в основном рассматривался вопрос о том, *может ли* быть разработан искусственный интеллект, но необходимо также посвятить время анализу вопроса, *должен ли* он все же быть разработан. Если последствия создания технологии искусственного интеллекта с большей вероятностью будут отрицательными, чем положительными, то люди, работающие в этой области, несут моральную ответственность, обязывающую их направить свои поиски в другие области. Непредвиденные отрицательные побочные эффекты стали следствием внедрения многих новых технологий: двигатели внутреннего сгорания послужили причиной загрязнения воздуха и повсеместного строительства дорог, даже в самых райских уголках; ядерные технологии стали причиной взрывов в Чернобыле и на острове Тринити Айленд и создали угрозу глобального разрушения. Все ученые и инженеры сталкиваются с этическими соображениями, которые они должны учитывать, выполняя свою работу, выбирая проекты, которые должны или не должны быть реализованы, а также способы осуществления этих проектов. На эту тему была даже написана книга *Ethics of Computing* [103]. Но искусственный интеллект, по-видимому, становится источником некоторых невиданных ранее проблем, в том числе перечисленных ниже, кроме, скажем, строительства мостов, которые падают под собственным весом.

- В результате автоматизации может увеличиться количество безработных.
- Может уменьшиться (или увеличиться) количество свободного времени, имеющегося в распоряжении людей.
- Люди могут потерять чувство собственной уникальности.
- Люди могут потерять некоторые из своих прав на личную жизнь.
- Использование систем искусственного интеллекта может привести к тому, что люди станут более безответственными.
- Успех искусственного интеллекта может стать началом конца человеческой расы.

Рассмотрим каждую из этих проблем по очереди.

- В результате автоматизации может увеличиться количество безработных. Современная индустриальная экономика стала полностью зависимой от применения компьютеров в целом и отдельных программ искусственного интеллекта в частности. Например, значительная часть экономики, особенно в

Соединенные Штаты, зависит от доступности потребительского кредита. Проверка заявок на выпуск кредитных карт, выдача разрешений на осуществление платежей и раскрытие мошеннических сделок, а также другие операции теперь выполняются программами искусственного интеллекта. Напрашивается вывод, будто из-за появления этих программ свои места потеряли тысячи служащих, но в действительности этих рабочих мест без применения программ искусственного интеллекта просто не было бы, поскольку в случае применения ручного труда стоимость этих операций была бы неприемлемой. До сих пор автоматизация с помощью технологии искусственного интеллекта неизменно создавала больше рабочих мест, чем устранила, к тому же приводила к появлению более интересных и высокооплачиваемых специальностей. Теперь, после того как канонической программой искусственного интеллекта стал “интеллектуальный агент”, предназначенный для помощи человеку, потеря работы становится еще менее вероятным последствием внедрения искусственного интеллекта по сравнению с той эпохой, когда все усилия специалистов по искусственному интеллекту сосредоточивались на создании “экспертных систем”, предназначенных для замены людей.

- Может уменьшиться (или увеличиться) количество свободного времени, имеющегося в распоряжении людей. Алвин Тоффлер в своей книге *Future Shock* [1510] указал: “Рабочая неделя с начала столетия сократилась на 50%. И никого не удивляет прогноз, что к 2000 году она будет сокращена еще наполовину”. Артур К. Кларк [265] писал, что “люди в 2001 году могут столкнуться с будущим, заполненным необычайной скучкой, когда главной проблемой в жизни станет принятие решения о том, какой из нескольких сотен телевизионных каналов выбрать для просмотра”. Единственным из этих прогнозов, который сбылся хоть в какой-то степени, стало количество телевизионных каналов [1453]. А вместо сокращения рабочего дня люди, занятые в отраслях промышленности, характеризующиеся интенсивным использованием знаний, стали чувствовать себя частью интегрированной компьютеризированной системы, которая работает 24 часа в сутки; чтобы успешно справляться со своими обязанностями, они вынуждены работать все больше и больше. В индустриальной экономике вознаграждения приблизительно пропорциональны затратами времени; увеличение продолжительности работы на 10% обычно приводит в среднем к увеличению доходов на 10%. А в информационной экономике, характеризующейся наличием широкополосной связи и упрощением тиражирования интеллектуальной собственности (которую Фрэнк и Кук [495] назвали “обществом, в котором победитель получает все”), самое большое вознаграждение приносит способность оказаться немного более успешным, чем конкурент; увеличение продолжительности работы на 10% может означать увеличение дохода на 100%. Поэтому каждый испытывает все возрастающий прессинг, который заставляет его работать все интенсивнее. Искусственный интеллект способствует увеличению темпов внедрения технологических инноваций и поэтому вносит свой вклад в эту общую тенденцию, но искусственный интеллект обещает также предоставить нам возможность снять с себя часть нагрузки и позволить нашим автоматизированным агентам хоть какое-то время выполнять работу за нас.

- Люди могут потерять чувство собственной уникальности. В своей книге *Computer Power and Human Reason* [1566] Вейценбаум, автор программы Eliza, указал на некоторые потенциальные угрозы, с которыми сталкивается общество в связи с развитием искусственного интеллекта. Одним из самых важных доводов Вейценбаума является то, что в результате исследований в области искусственного интеллекта кажется уже не такой невероятной идея о том, что люди представляют собой автоматы, а эта идея приводит к потере самостоятельности или даже человечности. Но авторы настоящей книги хотят отметить, что эта идея существовала задолго до появления искусственного интеллекта и восходит по меньшей мере к тем временам, когда вышла книга *L'Homme Machine* [875]. Авторы также отмечают, что люди пережили и другие покушения на чувство их уникальности: Коперник, автор книги *De Revolutionibus Orbium Coelestium* [294], убрал Землю из центра солнечной системы, а Дарвин, автор книги *Descent of Man* [326], поместил вид *Homo sapiens* на тот же уровень, где находятся все другие виды живых существ. Поэтому даже в случае его широкого и успешного наступления искусственный интеллект станет не большей угрозой для моральных устоев общества XXI века, чем была дарвиновская теория эволюции для XIX века.
- Люди могут потерять некоторые из своих прав на личную жизнь. Вейценбаум также указал, что развитие технологии распознавания речи может привести к широкому распространению средств прослушивания телефонных разговоров и поэтому к потере гражданских свобод. Он не предвидел, что когда-то в мире террористические угрозы станут настолько реальными, что изменят отношение людей к тому, с каким уровнем надзора они будут готовы согласиться, однако правильно понял, что искусственный интеллект обладает потенциалом к созданию средств надзора массового применения. Предсказание Вейценбаума вскоре может осуществиться<sup>6</sup>: правительство США рассматривает вопрос о внедрении системы Echelon, которая “состоит из сети пунктов прослушивания, антенных полей и радарных станций; система опирается на поддержку компьютеров, в которых используется языковой перевод, распознавание речи и поиск по ключевым словам для автоматического просеивания трафика, идущего по телефону, электронной почте, факсу и телексу”. Многие согласны с тем, что компьютеризация приводит к ущемлению прав на личную жизнь, — один из старших руководителей компании Sun Microsystems Скотт Макнили даже заявил: “У вас все равно нет никакой личной жизни. Забудьте о ней”. Другие с этим не согласны; например, судья Луис Брандейс писал еще в 1890 году: “Право на личную жизнь является самым всеобъемлющим из всех прав… ведь это — право оставаться самим собой”.
- Использование систем искусственного интеллекта может привести к тому, что люди станут более безответственными. В той атмосфере постоянной готовности к судебным разбирательствам, которая доминирует в Соединенных Штатах, большую важность приобретают вопросы правовой ответственности. Если терапевт принял на веру суждение медицинской экспертной системы в от-

<sup>6</sup> См. статью “Eavesdropping on Europe” (Применение средств подслушивания в Европе), Wired news, 9/30/1998, и процитированные в ней отчеты Европейского экономического сообщества.

ношении диагноза, то кто будет нести ответственность, если диагноз окажется неправильным? К счастью, теперь общепризнано, что нельзя рассматривать как пренебрежение служебными обязанностями выполнение терапевтом медицинских процедур, которые имеют высокую ожидаемую полезность, даже если фактические результаты оказались катастрофическими для пациента (отчасти такая смена взглядов обусловлена ростом влияния методов теории решений в медицине). Поэтому приведенный выше вопрос должен рассматриваться в такой формулировке: “Кто должен нести ответственность, если диагноз оказался неоправданным?” До сих пор суды исходили из того, что медицинские экспертные системы играют такую же роль, как медицинские учебники и справочники; терапевты обязаны понять ход рассуждений, лежащий в основе любого решения системы, и использовать свое собственное суждение для принятия решения о том, нужно ли следовать рекомендациям системы. Поэтому при проектировании медицинских экспертных систем в виде интеллектуальных агентов действия этих систем следует рассматривать не как непосредственно воздействующие на пациента, а как влияющие на поведение терапевта. А если экспертные системы когда-то будут надежно вырабатывать более точные диагнозы по сравнению с людьми, врачи могут стать юридически ответственными, если они не используют рекомендации экспертной системы. Такая предпосылка рассматривается в [528].

Приобретают также важное значение аналогичные вопросы, касающиеся использования интеллектуальных агентов в Internet. Например, достигнут определенный прогресс в части внедрения ограничений в интеллектуальные агенты, чтобы они не могли, допустим, повредить файлы других пользователей [1571]. Проблемы становятся еще более важными, когда речь идет о денежных сделках. Если финансовые операции выполняются интеллектуальным агентом “от чьего-то имени”, то кто будет отвечать за причиненные убытки? Будет ли существовать такая возможность, чтобы интеллектуальный агент сам имел активы и участвовал в электронных торгах от своего имени? Создается впечатление, что такие вопросы до сих пор еще полностью не изучены. Насколько известно авторам данной книги, еще ни одной программе не был придан правовой статус как индивидуума, способного участвовать в финансовых операциях; в настоящее время такое решение, по-видимому, было бы неблагоразумным. Кроме того, программы еще не рассматриваются как “водители”, когда речь идет о контроле за выполнением правил дорожного движения на реальных автомагистралях. По крайней мере, в Калифорнии не предусмотрено никаких юридических санкций, которые исключали бы возможность превышения автоматизированным транспортным средством скоростных пределов, хотя в случае аварии должен нести ответственность проектировщик механизма управления транспортным средством. Как и в случае технологии клонирования людей, законодателям еще предстоит включить новые разработки в правовое поле.

- Успех искусственного интеллекта может стать началом конца человеческой расы. Почти любая технология, попадая в злонамеренные руки, обнаруживает потенциальные возможности для причинения вреда, но когда речь идет об искусственном интеллекте и робототехнике, возникает новая проблема, связан-

ная с тем, что эти злонамеренные руки могут принадлежать самой технологии. Предупреждения о том, какую опасность несут роботы или роботизированные киборги-гуманоиды, вышедшие из-под контроля, стали сюжетом бесчисленных научно-фантастических произведений. К числу самых ранних примеров таких произведений относятся книга *Frankenstein, or the Modern Prometheus*<sup>7</sup> Мэри Шелли [1400] и пьеса *R.U.R* Карела Чапека (1921 год), в которых описано, как роботы завоевывают мир. В кинематографе этой теме посвящены фильм “The Terminator” (Терминатор), вышедший на экраны в 1984 году, в котором используются клише “о роботах, завоевающих мир” и сюжет о путешествии во времени, и фильм “The Matrix” (Матрица), вышедший в 1999 году, в котором объединены сюжеты “о роботах, завоевающих мир” и “о мозге, растущем в колбе”.

По-видимому, роботы являются главными героями такого большого количества художественных произведений о завоевании мира в основном из-за того, что они воплощают в себе неизвестное, точно так же, как ведьмы и приведения в сказках, которыми пугали людей в более ранние эпохи. Но действительно ли роботы создают более реальную угрозу, чем ведьмы и приведения? Если роботы правильно спроектированы как агенты, которые соблюдают интересы своего владельца, то они не создают таких угроз: роботы, развивающиеся на основе постоянного усовершенствования текущих проектов, будут служить своим хозяевам, а не бороться против них. Люди иногда используют свой интеллект в агрессивных формах, поскольку они обладают некоторыми врожденными агрессивными тенденциями, обусловленными естественным отбором. Но машины, созданные людьми, не нуждаются в чертах врожденной агрессивности, если только сами люди не решат, что они должны быть таковыми. С другой стороны, возможно, что компьютеры добьются своего рода победы над людьми, успешно служа и становясь незаменимыми, так же как и автомобили, которые в определенном смысле завоевали промышленно развитый мир. Рассмотрим один сценарий, заслуживающий дополнительного анализа. И. Дж. Гуд [576] писал в одном из своих произведений следующее.

Дадим определение сверхинтеллектуальной машине как способной намного превзойти во всех областях мыслительной деятельности любого человека, каким бы умным он не был. А поскольку проектирование машин является одним из таких направлений мыслительной деятельности, то сверхинтеллектуальная машина сможет проектировать еще более интеллектуальные машины; это, безусловно, приведет к “взрыву интеллектуальности” и интеллект человека останется далеко позади. Таким образом, первое изобретение сверхинтеллектуальной машины станет последним изобретением, которое когда-либо сможет сделать человек, и то, если только машина будет достаточно любезна, чтобы сообщить человеку, как держать ее под контролем.

Этому “взрыву интеллектуальности” профессор математики и автор научно-фантастических произведений Вернор Виндж присвоил другое название — **технологическое превосходство**; он писал: “В течение тридцати лет люди получат технологические средства для создания сверхчеловеческого интеллекта. Вскоре после этого эра людей закончится” [1542]. Гуд и Виндж (и многие другие) правильно отмечают, что в настоящее время кривая технологического прогресса

<sup>7</sup> Чарльз Бэббидж в молодости испытал сильные впечатления, читая книгу о приключениях Франкенштейна.

растет экспоненциально (достаточно вспомнить закон Мура). Однако прогноз, что эта кривая будет и дальше подниматься вверх, следуя законам почти бесконечного роста, был бы слишком смелым. До сих пор любая технология развивалась по S-образной кривой, согласно которой экспоненциальный рост в конечном итоге сходит на нет.

Виндж озабочен и обеспокоен грядущим технологическим превосходством, но другие специалисты в области компьютерных наук, занимающиеся прогнозами, радуются его наступлению. Ханс Моравек в своей книге *Robot: Mere Machine to Transcendent Mind* предсказывает, что роботы сравняются по интеллектуальности с человеком за 50 лет, после чего его превзойдут. В своей книге он пишет следующее.

Довольно скоро они станут способными вытеснить нас из жизни. Но я не очень встревожен такой возможностью, поскольку считаю, что будущие машины — это наше потомство, “порождение нашего разума”, созданное по нашему образу и подобию; это мы, но в более развитой форме. Как и биологические дети предыдущих поколений, они воплотят в себе лучшие надежды человечества на долгосрочное будущее. Это побуждает нас предоставить им все возможное содействие, а затем отойти в сторону, после того, как мы больше не в состоянии будем делать что-то полезное [1085].

Рей Курцвейл в своей книге *The Age of Spiritual Machines* [872] предсказывает, что к 2099 году наметится “мощная тенденция к слиянию человеческого мышления с миром машинного интеллекта, который был изначально создан родом человеческим. Больше не будет какого-либо четкого различия между людьми и компьютерами”. Возникло даже новое слово, **трансгуманизм**, для обозначения активного социального движения, которое охватывает тех, кто готовится к такому будущему. Достаточно сказать, что указанные вопросы становятся вызовом для большинства теоретиков морали, которые считают единственно возможным путем развития сохранение существования человека и самого человеческого рода.

Наконец, рассмотрим эту проблему с точки зрения робота. Если роботы обретут сознание, то трактовка их просто как “машин” (т.е. дальнейшее их принижение) может стать аморальной. Роботы сами должны также действовать морально — нам потребуется запрограммировать в них теорию, по которой они смогут судить, что хорошо и что плохо. Проблема прав и обязанностей роботов рассматривалась в произведениях авторов научной фантастики, начиная с Айзека Азимова [45]. В широко известном фильме “A.I.” [1451] Спилберга основой сюжета является рассказ Брайена Олдисса об интеллектуальном роботе, в программу которого была вложена вера в то, что он — человек, поэтому он не мог понять, почему же его когда-то неизбежно покинет владелица-мать. И в самом рассказе, использованном Спилбергом, и в его фильме приведены доводы в пользу того, что должно быть развернуто движение за гражданские права роботов.

## 26.4. РЕЗЮМЕ

В данной главе рассматривались приведенные ниже темы.

- Философы используют термин **слабый искусственный интеллект** для обозначения гипотезы о том, что машины могут обладать способностью действовать интеллектуально, и термин **сильный искусственный интеллект** — для обозна-

чения гипотезы, что такие машины можно рассматривать как действительно обладающие разумом (а не имитирующие разумную деятельность).

- Алан Тьюринг отверг как некорректный вопрос: “Могут ли машины мыслить?” и заменил его поведенческим тестом. Он сумел предугадать многие возражения против самой возможности появления мыслящих машин. В настоящее время исследователи искусственного интеллекта почти не уделяют тесту Тьюринга внимания, поскольку предпочитают работать над повышением производительности своих систем в решении практических задачах, а не над совершенствованием их способности имитировать людей.
- В настоящее время общепризнано, что психические состояния представляют собой состояния мозга.
- Споры за и против сильного искусственного интеллекта еще не закончились. Но лишь немногие из ведущих исследователей искусственного интеллекта считают, что итогом этих дебатов могут стать какие-либо существенные достижения.
- Проблема сознания продолжает оставаться нерешенной загадкой.
- Выявлено шесть потенциальных угроз обществу, создаваемых искусственным интеллектом и связанной с ним технологией. Авторы данной книги пришли к заключению, что некоторые из этих угроз либо являются маловероятными, либо ненамного отличаются по степени опасности от угроз, создаваемых другими, “неинтеллектуальными” технологиями. Лишь одна из эти угроз особо заслуживает дальнейшего анализа; она заключается в том, что с появлением сверхинтеллектуальных машин наступит такое будущее, которое весьма отличается от современности; это может нам не нравиться, но в данном вопросе у нас может не оказаться другого выбора. Указанные соображения неизбежно приводят к выводу, что мы должны тщательно взвешивать свои действия и в дальнейшем заняться анализом возможных последствий исследований искусственного интеллекта, от которых может зависеть будущее человеческой расы.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Природа разума всегда оставалась неизменной темой философских рассуждений от древних времен до настоящего времени. В своем произведении “Федон” Платон отдельно рассмотрел и отбросил идею о том, что разум может быть “настройщиком” или образцом организации частей тела, т.е. ту точку зрения, которая близка к взглядам приверженцев функционализма в современной философии разума. Вместо этого он решил, что разум должен быть воплощен в бессмертной, нематериальной душе, отделимой от тела и состоящей из другой субстанции, т.е. встал на позиции дуализма. Аристотель различал несколько видов душ (обозначая их греческим словом “ψυχή” — ψυχή) в живых существах, но некоторые из них он описывал в манере функционализма (дополнительные сведения о функционализме Аристотеля можно найти в [1151]).

Известным приверженцем дуалистических взглядов на человеческий разум был Декарт, но по иронии судьбы он оказал огромное историческое влияние на развитие

механицизма и материализма. Он явно рассматривал животных в качестве автоматов и предугадал предложенный Тьюрингом тест, написав в своей работе: “Нет ничего невероятного в том [что машина] может обладать способностью производить различные перестановки слов, чтобы дать достаточно осмысленный ответ на то, что сказано в ее присутствии, так же, как это способен сделать даже самый глупый из людей” [391]. Энергичная защита Декартом точки зрения на то, что животные являются автоматами, фактически позволила будущим философам скорее прийти к выводу о том, что люди также являются автоматами, даже несмотря на то, что сам он не сделал этого шага. В книге *L'Homme Machine* (Человек-машина) [875] Ламетри действительно приведено явное утверждение, что люди являются автоматами.

Приверженцы современной аналитической философии обычно признают материализм (часто в форме **теории идентичности** состояний мозга [39], [1215], согласно которой психические состояния идентичны состояниям мозга). Но в рамках материализма существуют серьезные разногласия по вопросам отношения к функционализму и применения машинной аналогии для человеческого разума, а также по вопросу о том, могут ли машины мыслить в буквальном понимании этого слова. На первых порах многие философы возражали против идей Тьюринга, изложенных в статье “*Computing Machinery and Intelligence*” [1520], например, Шривен [1374] пытался доказать, что бесмысленно даже говорить о том, что машины могут мыслить, на том основании, что в этом предположенииискажается сам смысл данного слова. Но к 1963 году Шривен отказался от этих взглядов; см. дополнение к перепечатке его статьи [28]. Специалист в области компьютерных наук Эдсгер Дейкстра сказал, что “вопрос о том, может ли компьютер мыслить, не более интересен, чем вопрос о том, может ли подводная лодка плавать”. А в [480] приведены доводы в пользу того, что тест Тьюринга не нужен для развития искусственного интеллекта.

**Функционализм** — это философия разума, которая зародилась под влиянием искусственного интеллекта наиболее естественным образом, поэтому критика функционализма часто принимает форму критики искусственного интеллекта (как в случае взглядов философа Сирла). Следуя классификации, используемой Блоком [138], мы можем различить несколько видов функционализма. **Теория функциональной спецификации** [923], [925] представляет собой одну из разновидностей теории идентичности состояний мозга, в которой выбираются состояния мозга, подлежащие идентификации с психическими состояниями на основе их функциональной роли. А **теория идентичности функционального состояния** [1246], [1248] более полно основана на машинной аналогии. В ней психические состояния идентифицируются не с физическими состояниями мозга, а с абстрактными вычислительными состояниями мозга, явно рассматриваемого как вычислительное устройство. Предполагается, что эти абстрактные состояния не зависят от конкретной физической композиции мозга, а это заставляет думать, что теория идентичности функциональных состояний представляет собой одну из форм дуализма!

И теория идентичности состояний мозга, и различные формы функционализма подвергаются атакам со стороны философов, утверждающих, что в этих взглядах не учитывается качественный аспект психических состояний или “аспект их подобия” [1110]. С другой стороны, Сирл сосредоточивается не на этом вопросе, а на том, что функционализм якобы не позволяет учитывать целенаправленность [1376], [1377], [1379]. В [259] приведено опровержение критических взглядов обоих этих типов.

Отличием ~~э~~ элиминирующего материализма [258], [1301] от всех других ведущих теорий в философии разума является то, что в нем не предпринимаются попытки обосновать правильность представлений “народной психологии” или обыденных представлений о разуме; вместо этого подобные взгляды отвергаются как ложные и предпринимается попытка заменить их чисто научной теорией разума. В принципе эта научная теория могла бы быть лучше развита в рамках классического искусственного интеллекта, но на практике приверженцы элиминирующего материализма склонны к проведению исследований в области неврологии и нейронных сетей [260] на том основании, что классический искусственный интеллект, особенно исследования в области “представления знаний”, описанные в главе 10, как правило, исходят из истинности “народной психологии”. Хотя точка зрения, основанная на “целенаправленной позиции” [387], может рассматриваться как соответствующая принципам функционализма, ее, возможно, следует вместо этого считать одной из форм элиминирующего материализма, поскольку предполагается, что принятие “целенаправленной позиции” не отражает каких-либо объективных свойств агента, по отношению к которым занята эта позиция. Следует также отметить такую возможность, что позиция элиминирующего материализма будет принята в отношении одних аспектов мыслительной деятельности, тогда как в отношении других будет принят какой-то другой подход. Например, Деннет [388] выдвигает гораздо более строгие требования по устранению из сферы анализа (в рамках элиминирующего материализма) понятия *качества*, чем понятия *целенаправленности*.

Ссылки на литературу, содержащую основные критические замечания в адрес слабого искусственного интеллекта, в основном приведены в данной главе. Хотя в эпоху, наступившую после бума в области нейронных сетей, стало модно высмеивать символические подходы, не все философы критически относятся к достижениям GOFAI. Напротив, некоторые из них являются его горячими сторонниками и даже практиками. Зенон Пилишин [1250] утверждает, что процесс познания можно лучше всего понять с помощью вычислительной модели, не только в принципе, но и в ходе проведения современных исследований. Он специально занимался опровержением критических замечаний Дрейфуса в адрес вычислительной модели человеческого познания [1249]. Гильберт Харман [621], анализируя процесс пересмотра взглядов, провел аналогии с исследованиями искусственного интеллекта на базе системы поддержки истинности. Майкл Братман применил свою модель человеческой психологии “убеждение–желание–намерение” [176] к исследованиям искусственного интеллекта по планированию [177]. Крайний приверженец строгого искусственного интеллекта, Арон Сломан [1432, с. xiii], даже назвал “расистскими” взгляды Джозефа Вейценбаума [1566] на то, что гипотетические интеллектуальные машины не следуют рассматривать как личности.

Философская литература по проблемам разума, мозга и близким к этому темам очень обширна; кроме того, эта литература часто является трудной для чтения теми, кто не обладает надлежащей подготовкой в области терминологии и используемых методов аргументирования. Исключительно авторитетным и очень полезным помощником в этом процессе может стать книга *Encyclopedia of Philosophy* (Энциклопедия философии) [431]. Более краткой и доступной книгой является *The Cambridge Dictionary of Philosophy* (Кембриджский словарь по философии) [48], но и в нем основные статьи (например, с описанием “философии разума”) все еще превышают по объему десятки страниц. Вопросы философии разума, а также биологии и психологии разума

рассматриваются в книге *MIT Encyclopedia of Cognitive Science* [1599]. К числу общих сборников статей о философии разума, в которых представлены различные точки зрения на проблемы, связанные с искусственным интеллектом, включая функционализм, относятся *Materialism and the Mind-Body Problem* [1309] и *Readings in the Philosophy of Psychology*, том 1 [138]. Биро и Шахан представили сборник статей [131], посвященный всем доводам за и против функционализма. К антологиям статей, в большей степени касающихся отношений между философией и искусственным интеллектом, относятся *Minds and Machines* [28], *Philosophical Perspectives in Artificial Intelligence* [1288], *Mind Design* [630] и *The Philosophy of Artificial Intelligence* [146]. Имеется также несколько общих введений в философский “вопрос об искусственном интеллекте” [145], [146], [293], [631]. Основным журналом, который публикует материалы философских и научных дебатов об искусственном интеллекте и неврологии, является *The Behavioral and Brain Sciences* (сокращенно *BBS*). Вопросы этики и ответственности в искусственном интеллекте рассматриваются в таких журналах, как *AI and Society*, *Law, Computers and Artificial Intelligence* и *Artificial Intelligence and Law*.

## УПРАЖНЕНИЯ

- 26.1.** Пройдите по составленному Тьюрингом списку предполагаемых действий, на которые “не способны” машины, и укажите, какие из них удалось осуществить, какие осуществимы в принципе с помощью программ и какие все еще находятся под вопросом, поскольку для них требуются сознательные психические состояния.
- 26.2.** Является ли опровержение довода с китайской комнатой убедительным доказательством того, что должным образом запрограммированные компьютеры имеют психические состояния? Обязательно ли принятие этого довода означает согласие с тем, что компьютеры не могут иметь психических состояний?
- 26.3.** Пользуясь доводом с протезом мозга, важно предусмотреть способность восстановить мозг субъекта до нормального состояния, так, чтобы внешне он вел себя, будто над ним не проводилась какая-либо операция. Будет ли обоснованным возражение скептика о том, что для этого потребуется обновление нейрофизиологических свойств нейронов, которые относятся к сознательному опыту, в отличие от тех свойств, которые касаются функционального поведения нейронов?
- 26.4.** Найдите и проанализируйте в популярном издании одну или несколько статей на ту тему, что создание искусственного интеллекта является невозможным.
- 26.5.** Попытайтесь составить определения терминов “интеллект”, “мышление” и “сознание”. Проанализируйте некоторые возможные возражения против ваших определений.
- 26.6.** Проанализируйте потенциальные угрозы обществу со стороны технологии искусственного интеллекта. Какие угрозы являются наиболее серьезными и как им можно противостоять? В каком соотношении они находятся с потенциальными выгодами?

- 26.7.** Каковыми являются потенциальные угрозы со стороны технологии искусственного интеллекта в сравнении с угрозами со стороны других компьютерных технологий, а также со стороны ядерных, био- и нано- технологий?
- 26.8.** Одни критики искусственного интеллекта утверждают, что его создать невозможно, а другие считают, что очень даже возможно и что сверхинтеллектуальные машины представляют собой угрозу. Какое из этих утверждений вы считаете наиболее вероятным? Будут ли противоречивыми взгляды тех, кто занимает одновременно обе позиции?

# 27 НАСТОЯЩЕЕ И БУДУЩЕЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

*В данной главе мы подводим итог тому, где мы находимся и куда движемся; это следует сделать, прежде чем приступить к дальнейшему изложению ее темы.*

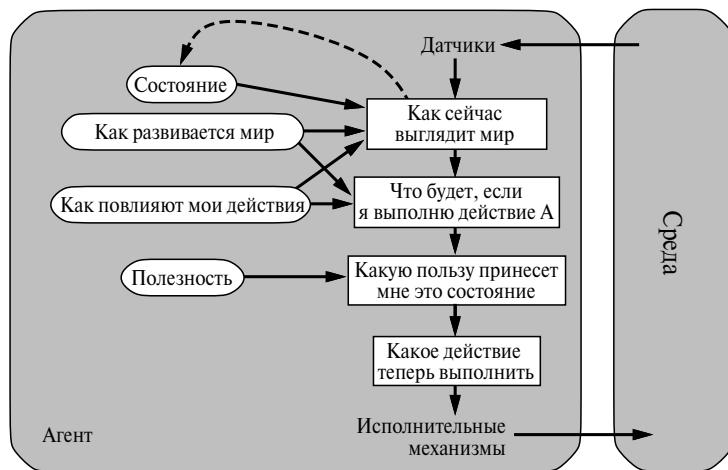
В части I предложен единый подход к проблематике искусственного интеллекта — как к проектированию рациональных агентов. В ней было показано, что сложность задачи проектирования зависит от того, какие акты восприятия и действия доступны для агента, каким целям должно удовлетворять поведение агента, а также от характера среды. Возможно применение целого ряда различных проектов агентов, начиная от простых рефлексных агентов и заканчивая агентами, основанными на знаниях, полностью способными к самостоятельному функционированию. Более того, компоненты этих проектов могут иметь целый ряд различных реализаций, таких как логические, вероятностные или “нейронные”. А в главах, которые следуют за этой частью, представлены принципы, на основании которых действуют эти компоненты.

Как описано выше, в последнее время достигнуты поразительные успехи в научном понимании проблемы и в обеспечении технологических возможностей как с точки зрения разработки проектов, так и с точки зрения реализации компонентов агентов. В данной главе мы отвлечемся от деталей и попытаемся найти ответ на вопрос: «*Приведет ли весь этот прогресс к созданию интеллектуального агента общего назначения, способного функционировать в самых различных вариантах среды?*» В разделе 27.1 рассматриваются компоненты интеллектуального агента для определения того, что известно и чего еще недостает. В разделе 27.2 решается такая же задача применительно к общей архитектуре агента. В разделе 27.3 предпринимается попытка найти ответ на вопрос о том, является ли, прежде всего, правильной сама цель создания “проекта рационального агента” (ответ на данный вопрос должен быть таков: “Фактически дело обстоит иначе, но на данный момент эта цель являет-

ся вполне приемлемой"). Наконец, в разделе 27.4 показано, к каким последствиям приведет достижение нами успеха в своих начинаниях.

## 27.1. КОМПОНЕНТЫ АГЕНТА

В главе 2 представлены разнообразные проекты агентов и их компоненты. Но в настоящей главе, для того чтобы можно было сузить рамки обсуждения, будет рассматриваться проект агента, действующего с учетом полезности, который еще раз показан на рис. 27.1. Это — наиболее общий из рассматриваемых нами проектов агентов; в этой главе речь также пойдет о его дополнении средствами обучения, как показано на рис. 2.7.



*Рис. 27.1. Проект, основанный на модели агента, действующего с учетом полезности, который впервые представлен на рис. 2.6*

- **Взаимодействие со средой с помощью датчиков и исполнительных механизмов.** Этот аспект оставался очевидным для всех слабым пунктом на протяжении большей части истории развития искусственного интеллекта. Если не считать небольшого числа исключений, вызвавших всеобщее восхищение, системы искусственного интеллекта создавались таким образом, что людям приходилось вручную подавать в них входные данные и интерпретировать выходные, поскольку робототехнические системы, предназначенные для решения задач низкого уровня, на которые опирались бы высокуюровневые компоненты формирования рассуждений и планирования, в основном отсутствовали. Такая ситуация отчасти была обусловлена тем, что для обеспечения функционирования настоящих роботов даже в самой узкой области требовались существенные финансовые расходы и большие затраты труда проектировщиков. Такая ситуация быстро изменяется в последние годы в связи с появлением готовых программируемых роботов, таких как роботы с четырьмя опорными конечностями (см. рис. 25.4, б). Эти роботы, в свою очередь, созданы благодаря появлению небольших, недорогих телекамер CCD (Charge Coupled Device — прибор с за-

рядовой связью) с высоким разрешением, а также компактных, надежных электрических приводов. Технология MEMS (Micro-ElectroMechanical System — микроскопические электромеханические системы) позволила создать миниатюрные акселерометры и гироскопы, а в настоящее время в ее рамках создаются исполнительные механизмы, способные, например, приводить в действие искусственное летающее насекомое. (Существует также возможность объединять миллионы исполнительных механизмов MEMS для получения очень мощных макроскопических исполнительных механизмов.) Поэтому, что касается вариантов физической среды, то больше нет реальных причин, оправдывающих то положение, в котором находятся системы искусственного интеллекта. Кроме того, стала доступной полностью новая среда — Internet.

- **Слежение за состоянием мира.** Это — одна из основных способностей, которой должен обладать интеллектуальный агент. Для этого требуется и восприятие, и обновление внутренних представлений. В главе 7 описаны методы слежения за миром, представленные в форме пропозициональной логики; в главе 10 они расширены до логики первого порядка, а в главе 15 представлены алгоритмы **фильтрации** для слежения за неопределенными вариантами среды. Эти инструментальные средства фильтрации вступают в действие, когда приходится сталкиваться с реальными (поэтому далекими от идеала) результатами восприятия. Современные алгоритмы фильтрации и восприятия могут комбинироваться для успешного выполнения заданий по составлению сообщений в виде предикатов низкого уровня, таких как “на столе стоит чашка”, но еще многое предстоит сделать, прежде чем с помощью этих алгоритмов можно будет составить отчет, например, о том, что “доктор Рассел пьет чай с доктором Норвигом”. Еще одна проблема состоит в том, что алгоритмы приближенной фильтрации, хотя и могут действовать в весьма обширной среде, остаются по сути пропозициональными, поэтому, как и пропозициональная логика, не позволяют явно представлять объекты и отношения. В главе 14 описано, как можно применить в сочетании теорию вероятностей и логику первого порядка для решения этой задачи; можно рассчитывать на то, что применение этих идей для слежения за сложными вариантами среды со временем позволит добиться огромных преимуществ. Кстати, как только речь заходит об объектах в неопределенной среде, нам приходится сталкиваться с **неопределенностью идентичности**, поскольку часто неизвестно, не потерян ли из виду тот объект, за которым мы начинали следить. Эта проблема в системах искусственного интеллекта, основанных на логике, почти всегда игнорировалась, поскольку в основном предполагалось, что результаты восприятия включают константные символы, которые однозначно обозначают те или иные объекты.
- **Проектирование, оценка и выбор будущих способов действий.** При решении этой задачи требования к представлению основных знаний остаются такими же, как и при решении задачи слежения за миром; трудности состоят главным образом в том, что приходится сталкиваться с проявлениями действий (например, связанных с проведением беседы или совместного чаепития), которые в конечном итоге состоят из тысяч или миллионов примитивных шагов, выполняемых реальным агентом. Вообще говоря, люди осуществляют такое сложное поведение исключительно благодаря тому, что действуют в рамках

**иерархической структуры** поведенческих актов. В некоторых из алгоритмов планирования, приведенных в главе 12, используются иерархические представления и представления в логике первого порядка, позволяющие справляться с проблемами реальных масштабов; с другой стороны, в алгоритмах принятия решений в условиях неопределенности, приведенных в главе 17, по существу используются такие же идеи, как и в алгоритмах поиска с учетом состояния, рассматриваемых в главе 3. В этой области необходимо выполнить еще очень большой объем работы, возможно, на основе новейших достижений в области **иерархического обучения с подкреплением**.

- **Полезность как способ выражения предпочтений.** Вообще говоря, принцип, согласно которому рациональные решения должны быть основаны на максимизации ожидаемой полезности, является полностью общим и позволяет избежать многих проблем, связанных с подходами, основанными исключительно на достижении цели, таких как конфликтующие цели и ненадежные результаты. Однако до сих пор еще очень мало сделано в области создания реальных функций полезности. Достаточно представить себе, например, в какой сложной сети взаимодействующих предпочтений должен разбираться агент, действующий в качестве ассистента-делопроизводителя для чиновника. Как оказалось, задача декомпозиции предпочтений по сложным состояниям, подобная тому, как осуществляется декомпозиция убеждений по сложным состояниям в байесовских сетях, является весьма трудноразрешимой. Одна из причин этого может состоять в том, что распределение предпочтений по состояниям фактически компилируется из предпочтений, распределенных по историям состояний, которые описываются с помощью **функций вознаграждения** (см. главу 17). Даже если функция вознаграждения является простой, соответствующая функция полезности может оказаться очень сложной. Это означает, что мы должны рассматривать задачу инженерии знаний для функций вознаграждения, которая должна стать способом информирования разрабатываемых агентов о том, какие к ним предъявляются требования, как очень серьезную.
- **Обучение.** В главах 18–20 описано, как может быть сформулирована задача обучения агента в виде задачи определения с помощью индуктивного обучения (контролируемого, неконтролируемого или основанного на подкреплении) тех функций, которые лежат в основе различных компонентов агента. Были разработаны очень мощные логические и статистические методы, позволяющие справляться с весьма значительными проблемами, часто достигающие или превосходящие возможности человека по идентификации предсказательных шаблонов, определенных в заданном словаре. С другой стороны, в области машинного обучения достигнуты лишь весьма небольшие успехи в решении важной проблемы формирования новых представлений на уровнях абстракции, более высоких по сравнению с входным словарем. Например, как может автономный робот выработать полезные предикаты, такие как *Office* и *Cafe*, если они не будут предоставлены ему разработчиком? Аналогичные соображения распространяются и на проблему определения с помощью обучения способа поведения; например, участие в чаепитии *HavingACupOfTea* — это важное действие высокого уровня, но как ввести его в библиотеку действий, которая первоначально содержит гораздо более

простые действия, такие как *RaiseArm* (Поднять руку) и *Swallow* (Сделать глоток)? Если мы не поймем специфику таких проблем, то столкнемся с утомительной задачей построения больших баз обыденных знаний вручную.

## 27.2. АРХИТЕКТУРЫ АГЕНТОВ

Возникает резонный вопрос: “Какие архитектуры агентов, описанные в главе 2, должны использоваться в агентах?” Ответом является: “Все архитектуры!” Выше было показано, что рефлексные реакции требуются в тех ситуациях, в которых существенным является фактор времени, а с другой стороны, рассуждения, основанные на знаниях, позволяют агенту планировать наперед. Полноценный агент должен быть способным выполнять и то и другое с использованием **гибридной архитектуры**. Одним из важных свойств гибридных архитектур является то, что границы между различными компонентами, обеспечивающими принятие решений, не постоянны. Например, в процессе **компиляции** декларативная информация, полученная на уровне формирования рассуждений, последовательно преобразуется во все более эффективные представления, что позволяет в конечном итоге достичь рефлексного уровня, как показано на рис. 27.2. (В этом состоит цель обучения на основе объяснения, как описано в главе 19.) Точно такую же структуру имеют такие архитектуры агентов, как Soar [880] и Theo [1063]. После решения каждой задачи с помощью явного формирования рассуждений эти агенты сохраняют обобщенную версию полученного решения для его использования в рефлексном компоненте. Менее изученной проблемой является проблема осуществления процесса, обратного указанному, — после изменения среды рефлексы, усвоенные в результате обучения, могут оказаться больше не подходящими, и агенту может потребоваться вернуться на уровень формирования рассуждений, для того чтобы выработать новые способы поведения.

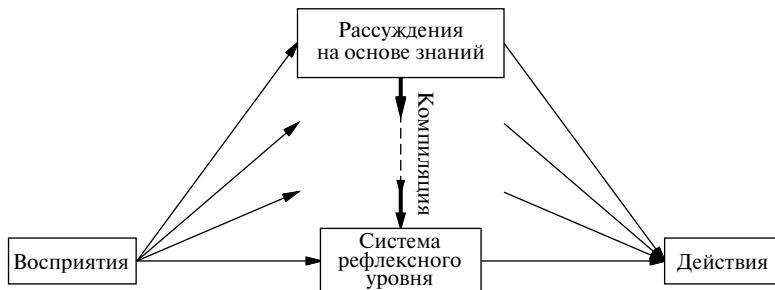


Рис. 27.2. Компиляция служит для преобразования результатов принятия решений на основе рассуждений в более эффективные рефлексивные механизмы

Агентам могут также потребоваться способы, позволяющие управлять своими собственными процессами формирования рассуждений. Они должны быть способными прекратить размышления, когда потребуются действия, а также должны умело использовать время, отведенное на рассуждения, чтобы выполнить наиболее продуктивные вычисления. Например, агент-водитель такси, который обнаружил впереди картину дорожного происшествия, должен решить за долю секунды, следует ли ему затормозить или объехать то место, где случилось происшествие. Кроме того,

данный агент должен также потратить лишь долю секунды на размышление о наиболее важных в этой ситуации вопросах, например, нет ли движения на полосах слева и справа и нет ли непосредственно сзади него большого грузовика, но не задумываться над тем, что резкий маневр увеличит износ и стирание шин автомобиля или что ему давно нужно было найти очередного пассажира. Исследование таких проблем осуществляется главным образом в рамках направления **искусственного интеллекта реального времени**. По мере того как системы искусственного интеллекта проникают во все более сложные проблемные области, все решаемые задачи становятся задачами реального времени, поскольку агенту никогда больше не отводится достаточно времени для точного решения задачи принятия решений.

Очевидно, что становится насущной потребность в методах, которые позволяют действовать в более общих ситуациях принятия решений. В последние годы появились два перспективных метода. В первом из них предусматривается использование **алгоритмов с отсечением по времени** [357], [682]. Алгоритмом с отсечением по времени называется алгоритм, качество выходных данных которого неизменно улучшается во времени, поэтому он всегда готов предоставить приемлемое решение, когда бы ни была прервана его работа. Такие алгоритмы действуют под управлением метауровневой процедуры принятия решений, которая оценивает, стоит ли выполнять дальнейшие вычисления. Простым примером алгоритма с отсечением по времени является поиск с итеративным углублением в задачах ведения игр. Могут также быть созданы более сложные системы, состоящие из многих таких алгоритмов, действующих вместе [1647]. Вторым методом являются **метарассуждения на основе теории решений** [683], [687], [1332]. В этом методе применяется теория ценности информации (см. главу 16) для выбора вычислений. Ценность вычислений зависит и от их стоимости (с точки зрения того, что действие не проводится, пока они осуществляются) и от их преимуществ (измеряемых с учетом того, насколько повысилось качество решения). Методы формирования метарассуждений могут использоваться для проектирования лучших алгоритмов поиска и для обеспечения гарантий того, что алгоритмы будут обладать вневременным свойством. Безусловно, подход на основе метарассуждений является дорогостоящим, а методы компиляции могут применяться таким образом, чтобы издержки были малы по сравнению со стоимостями контролируемых вычислений.

Тем не менее метарассуждения представляют собой лишь один из аспектов общей **рефлексивной архитектуры**, т.е. архитектуры, позволяющей формировать рассуждения о вычислительных сущностях и действиях, возникающих в самой архитектуре. Теоретическую основу для рефлексивных архитектур можно заложить, определив совместное пространство состояний, складывающееся из состояний среды и вычислительного состояния самого агента. Могут быть спроектированы алгоритмы принятия решений и обучения, которые применяются к этому совместному пространству состояний и поэтому способствуют реализации и совершенствованию вычислительной деятельности агента. Мы надеемся, что в конечном итоге такие алгоритмы, предназначенные для решения узко конкретных задач, как альфа-бета поиск и обратный логический вывод, исчезнут из систем искусственного интеллекта и будут заменены общими методами, которые направляют вычисления агента в сторону эффективного формирования высококачественных решений.

### 27.3. ОЦЕНКА ПРАВИЛЬНОСТИ ВЫБРАННОГО НАПРАВЛЕНИЯ

В предыдущем разделе были перечислены многие достижения и многие возможности для дальнейшего прогресса. Но в каком направлении идет все это развитие? Дрейфус [415] проводит аналогию с попыткой достать до луны, взираясь на дерево; в ходе этого наблюдается постоянный прогресс до тех пор, пока не будет достигнута вершина дерева. В этом разделе мы рассмотрим, напоминает ли текущий путь развития искусственного интеллекта подъем по стволу дерева или взлет ракеты.

В главе 1 было указано, что наша цель состоит в создании агентов, которые действуют рационально, но в той же главе было также указано следующее:

...задача достижения идеальной рациональности, при которой всегда выполняются правильные действия, не осуществима. Дело в том, что при этом предъявляются слишком высокие требования к вычислительным ресурсам. Но в основной части данной книги применяется рабочая гипотеза, согласно которой идеальная рациональность является хорошей отправной точкой для анализа.

Теперь настало время определить, в чем именно состоит цель искусственного интеллекта. Мы стремимся создавать агентов, но какой спецификацией следует при этом руководствоваться? Ниже описаны четыре возможных варианта.

- **↗ Идеальная рациональность.** Идеально рациональный агент в каждое мгновение действует таким образом, чтобы максимизировать свою ожидаемую полезность с использованием информации, полученной им из среды. Но было показано, что вычисления, необходимые для достижения идеальной рациональности, в большинстве вариантов среды требуют слишком больших затрат времени, поэтому задача обеспечения идеальной рациональности не является реалистичной.
- **↗ Вычислительная рациональность.** Это понятие рациональности использовалось нами неявно при проектировании логических агентов и агентов, основанных на принятии решений. Вычислительно рациональный агент в конечном итоге возвращает то, что рассматривалось как рациональный выбор в начале этапа формирования им рассуждений. Такое свойство может представлять интерес, только если система используется лишь в демонстрационных целях, а в большинстве вариантов среды правильный ответ теряет свою ценность, если он не был своевременным. На практике проектировщики систем искусственного интеллекта вынуждены искать компромисс между требованиями по обеспечению качества решений и требованиями по уменьшению затрат времени на получение приемлемой общей производительности; к сожалению, теоретические основы вычислительной рациональности не позволяют предложить достаточно обоснованного способа выработки таких компромиссов.
- **↗ Ограниченная рациональность.** Герберт Саймон [1415] отверг идею идеальной (или даже приближенно идеальной) рациональности и предложил использовать понятие ограниченной рациональности — описательную теорию принятия решений реальными агентами. Ниже приведена цитата из его работы.

Способность человеческого разума формулировать и решать сложные задачи слишком мала по сравнению с масштабами задач, для которых требуется искать решение, чтобы осуществлять объективно рациональное поведение в реальном мире, или хотя бы добиваться приемлемого приближения к такой объективной рациональности.

Саймон выдвинул предложение, что принцип ограниченной рациональности главным образом основан на принципе **непрятательности** (satisficing), т.е. на том, что проведение рассуждений следует осуществлять достаточно долго лишь для того, чтобы предложить “вполне приемлемый” ответ. За указанную выше работу Саймон получил Нобелевскую премию по экономике, а позднее выпустил книгу, в которой подробно осветил все свои идеи [1418]. По-видимому, понятие непрятательности может служить полезной моделью человеческого поведения во многих случаях. Но оно не является формальной спецификацией для интеллектуальных агентов, поскольку в теории Саймона не дано определение “вполне приемлемого” ответа. Кроме того, принцип непрятательности, по-видимому, является одним из широкого спектра методов, используемых для осуществления действий в условиях ограниченных ресурсов.

- **Ограниченнная оптимальность** (Bounded Optimality — BO). Ограниченно оптимальный агент действует настолько хорошо, насколько это возможно, с учетом его вычислительных ресурсов. Это означает, что ожидаемая полезность программы агента для ограниченного оптимального агента является по меньшей мере такой же высокой, как и ожидаемая полезность любой другой агентской программы, работающей на том же компьютере.

По-видимому, наилучшие перспективы создания мощного теоретического фундамента для искусственного интеллекта открывает только одна из этих четырех возможностей — ограниченная оптимальность. Преимущество связанного с ней подхода состоит в том, что он является осуществимым, — всегда можно найти по меньшей мере одну наилучшую программу, а таким свойством не обладает подход с идеальной рациональностью. Ограниченно оптимальные агенты действительно применимы в реальном мире, тогда как вычислительно рациональные агенты обычно неприменимы, а агенты, действующие по принципу непрятательности, могут оказаться применимыми или нет, в зависимости от их собственных конструктивных особенностей.

Традиционным подходом в искусственном интеллекте было то, что нужно начинать с вычислительной рациональности, а затем вырабатывать компромиссы с учетом ресурсных ограничений. Если проблемы, связанные с применением ограничений, не столь существенны, то можно надеяться на создание окончательного проекта, аналогичного проекту ограниченно оптимального агента. Но, по мере того как ресурсные ограничения становятся все более важными (например, по мере усложнения среды), может оказаться так, что два проекта станут весьма несхожими. А в теории ограниченной оптимальности эти ограничения могут учитываться в рамках целостного подхода.

До сих пор объем знаний в области ограниченной оптимальности остается не таким уж значительным. Известно, что могут быть созданы ограниченно оптимальные программы для очень простых машин и для довольно лимитированных вариантов среды [445], [1330], но еще нет полного представления о том, какими должны быть программы BO для больших компьютеров общего назначения, применяемых в сложных вариантах среды. Если теория ограниченной оптимальности будет носить конструктивный характер, то можно рассчитывать на получение проектов ограниченно оптимальных программ, которые не слишком сильно зависят от устройства используемого компьютера. Научные исследования стали бы весьма затруднительными, если бы увеличение объема памяти гигабайтового компьютера на несколько килобайтов привело к существенному изменению программы BO. Одним из способов обеспечения того, чтобы это не могло

случиться, может служить небольшое ослабление критериев ограниченной оптимальности. По аналогии с понятием асимптотической сложности (приложение А) можно определить понятие **асимптотической ограниченной оптимальности** (Asymptotic Bounded Optimality — ABO), как описано ниже [1329]. Предположим, что программа  $P$  является ограниченно оптимальной для компьютера  $M$  в классе вариантов среды  $E$ , тогда как сложность вариантов среды в  $E$  не ограничена. В таком случае программа  $P'$  обладает свойством АВО для  $M$  в  $E$ , если она может превзойти по производительности программу  $P$ , работая на компьютере  $kM$ , который в  $k$  раз быстрее (или крупнее) по сравнению с  $M$ . За исключением предельных значений  $k$  было бы достаточно иметь программу, обладающую свойством АВО, для нетривиальной среды в нетривиальной архитектуре. Было бы мало смысла затрачивать невероятные усилия на поиск программ ВО, а не АВО, поскольку все равно размеры и скорость доступных компьютеров увеличиваются на постоянный коэффициент через фиксированные промежутки времени, в связи с появлением каждого нового поколения этих устройств.

Можно рискнуть предположить, что программы ВО или АВО для мощных компьютеров, действующих в сложных вариантах среды, не обязательно должны иметь простую, изящную структуру. Выше уже было показано, что для искусственного интеллекта общего назначения требуются некоторые рефлексивные способности и некоторые способности к формированию рассуждений, целый ряд форм представления знаний и принятия решений, механизмы обучения и компиляции для всех этих форм, методы управления процессом формирования рассуждений и большой запас знаний в данной конкретной области. Ограниченно оптимальный агент должен адаптироваться к той среде, в которой он находится сам, с тем чтобы в конечном итоге его внутренняя организация соответствовала возможностям оптимизации, характерным для данной конкретной среды. Можно рассчитывать лишь на указанную возможность, а такой ход развития аналогичен пути, по которому развивались гоночные автомобили с ограничениями на мощность, пока наконец не были созданы чрезвычайно сложные, но весьма эффективные проекты. По мнению авторов, наука искусственного интеллекта, основанная на понятии ограниченной оптимальности, будет способствовать интенсивному исследованию процессов, позволяющих путем последовательных итераций создавать агентские программы с ограниченной оптимальностью и, возможно, меньше сосредоточиваться на анализе того, как именно устроены создаваемые при этом программы, пусть даже не такие изящные.

Подводя итог, можно отметить, что проведение разработки понятия ограниченной оптимальности было предложено в качестве формальной задачи для исследований по искусственноому интеллекту, которая не только хорошо определена, но и осуществима. Ограниченная оптимальность определяет оптимальные программы, но не оптимальные действия. А действия в конечном итогерабатываются программами и с помощью программ, которые полностью зависят от замысла проектировщика.

## 27.4. ПЕРСПЕКТИВЫ РАЗВИТИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Дэвид Лодж в своем романе *Small World* [943] об академическом мире литературной критики описал сцену, в которой главный герой вызывает замешательство среди

группы выдающихся, но несогласных друг с другом теоретиков литературы, задав им следующий вопрос: “Что было бы, если бы вы оказались правы?” Ни один из теоретиков, по-видимому, еще не задумывался над этим вопросом раньше, поскольку ведение споров по поводу неопровергимых теорий — бессмысленное занятие. Аналогичное замешательство можно вызвать, задав исследователям в области искусственного интеллекта вопрос: “Что было бы, если бы вам удалось достичь своей цели?” Ведь искусственный интеллект демонстрирует замечательные достижения, а интеллектуальные компьютеры, безусловно, более полезны, чем неинтеллектуальные компьютеры, так о чём же беспокоиться?

Как было указано в разделе 26.3, необходимо рассмотреть некоторые этические проблемы. Интеллектуальные компьютеры являются более мощными, но будет ли эта мощь использоваться во благо или во зло? Те, кто посвящают свою жизнь разработкам в области искусственного интеллекта, ответственны за то, чтобы влияние их работы было положительным. Широта этого влияния зависит от степени успеха искусственного интеллекта. Даже первые скромные успехи в области искусственного интеллекта повлияли на то, как осуществляются преподавание компьютерных наук [1459] и разработка программного обеспечения. Благодаря профессиональному интеллекту удалось создать принципиально новые приложения, такие как системы распознавания речи, системы управления запасами, интеллектуальные системы наружного наблюдения, роботы и машины поиска.

Авторы считают, что достижение в искусственном интеллекте успехов среднего уровня окажет влияние на повседневную жизнь всех слоев населения во всем мире. До сих пор такого рода всепроникающее воздействие на общество смогли оказать лишь компьютеризированные сети связи, такие как сеть сотовой телефонной связи и Internet, а искусственный интеллект оставался в стороне. Вполне можно представить себе, что действительно полезные персональные ассистенты для офиса или дома окажут большое положительное воздействие на повышение качества повседневной жизни, хотя они в краткосрочной перспективе и могут вызвать некоторые экономические неурядицы. Кроме того, технологические возможности, открывающиеся на этом уровне, могут быть также применены для создания автономного оружия, появление которого многие считают нежелательным.

Наконец, кажется вполне вероятным, что крупномасштабный успех в создании искусственного интеллекта (появление интеллекта на уровне человека и превосходящего его) повлияет на существование большинства представителей рода человеческого. Изменится сам характер нашей работы и развлечений, так же как и наши представления об интеллекте, сознании и будущей судьбе человечества. На этом уровне системы искусственного интеллекта могут создать более непосредственную угрозу самоопределению, свободе и даже выживанию людей. По этим причинам нельзя рассматривать исследования в области искусственного интеллекта в отрыве от их этических последствий.

Каким мы видим будущее? Авторы научно-фантастических романов, по-видимому, чаще публикуют пессимистические, а не оптимистические сценарии грядущего, возможно, потому, что это позволяет сочинять более увлекательные сюжеты. Но создается впечатление, что искусственный интеллект пока развивается по тому же принципу, как и другие революционные технологии (типографское дело, инженерное оборудование, воздухоплавание, телефония и т.д.), отрицательные последствия внедрения которых перевешиваются положительными результатами.

В заключение следует отметить, что за свою короткую историю искусственный интеллект добился существенного прогресса, но последнее утверждение из эссе Алана Тьюринга *Computing Machinery and Intelligence* продолжает оставаться неоспоримым и в наши дни.

Мы способны заглянуть вперед лишь на очень короткое расстояние, но все равно можем увидеть, как много еще предстоит сделать.

# А МАТЕМАТИЧЕСКИЕ ОСНОВЫ

## А.1. АНАЛИЗ СЛОЖНОСТИ И СИСТЕМА ОБОЗНАЧЕНИЙ $O()$

Специалистам в области компьютерных наук часто приходится решать задачу сравнения алгоритмов для определения того, насколько быстро они действуют или сколько памяти для них требуется. Для решения этой задачи предусмотрены два подхода. Первым из них является **применение эталонных тестов** — прогон реализующих алгоритмы программ на компьютере и измерение быстродействия в секундах и потребления памяти в байтах. Верно, что в конечном итоге нас действительно интересуют именно такие практические характеристики, но эталонное тестирование может оказаться неприемлемым потому, что оно слишком узко направлено, — в нем измеряется производительность конкретной программы, написанной на конкретном языке, функционирующей на конкретном компьютере с конкретным компилятором и с конкретными входными данными. К тому же на основании единственного результата, полученного с помощью эталонного тестирования, трудно предсказать, насколько успешно этот алгоритм будет действовать в случае использования другого компилятора, компьютера или набора данных.

### Асимптотический анализ

Второй подход опирается на математический **анализ алгоритмов**, независимый от конкретной реализации и входных данных. Рассмотрим этот подход на приведенном ниже примере программы, которая вычисляет сумму последовательности чисел (листинг А.1).

#### Листинг А.1. Программа вычисления суммы последовательности чисел

```
function Summation(sequence) returns сумма sum
    sum ← 0
    for i ← 1 to Length(sequence)
        sum ← sum + sequence[i]
    return sum
```

Первый этап анализа состоит в том, что создается определенное абстрактное представление входных данных, позволяющее найти какой-то параметр (или параметры), который характеризует объем входных данных. В рассматриваемом примере объем входных данных можно охарактеризовать с помощью такого параметра, как длина последовательности, которая будет обозначаться как  $n$ . На втором этапе необходимо определить абстрактное представление реализации и найти какой-то критерий, отражающий продолжительность прогона алгоритма, но не привязанный к конкретному компилятору или компьютеру. Применимально к программе *Summation* этим критерием может служить количество строк выполняемого кода; кроме того, данный критерий может быть более детализированным и измеряющим количество сложений, присваиваний, обращений к элементам массивов, а также ветвлений, выполняемых в этом алгоритме. В любом случае будет получена характеристика общего количества шагов, выполняемых алгоритмом, как функция от объема входных данных. Обозначим эту характеристику как  $T(n)$ . Если за основу берется количество строк кода, то в данном примере  $T(n) = 2n + 2$ .

Если бы все программы были такими же простыми, как *Summation*, то область анализа алгоритмов не заслуживала бы названия научной. Но исследования в этой области существенно усложняются из-за наличия двух проблем. Первая проблема заключается в том, что редко удается найти параметр, подобный  $T(n)$ , который полностью характеризует количество шагов, выполняемых в алгоритме. Вместо этого обычно можно самое большое вычислить этот показатель для наихудшего случая  $T_{\text{worst}}(n)$  или для среднего случая  $T_{\text{avg}}(n)$ . А для вычисления среднего требуется, чтобы аналитик принял какие-то обоснованные предположения по распределению наборов входных данных.

Вторая проблема состоит в том, что алгоритмы обычно не поддаются точному анализу. В этом случае приходится прибегать к аппроксимации и использовать, например, такую формулировку, что быстродействие алгоритма *Summation* характеризуется величиной  $O(n)$ . Это означает, что быстродействие алгоритма измеряется величиной, пропорциональной  $n$ , возможно, за исключением нескольких небольших значений  $n$ . Более формально это определение можно представить с помощью следующей формулы:

$$T(n) \text{ равно } O(f(n)), \text{ если } T(n) \leq kf(n) \text{ для некоторого } k, \text{ при всех } n > n_0$$

Перейдя к использованию системы обозначений  $O()$ , мы получаем возможность воспользоваться так называемым **асимптотическим анализом**. А в рамках этого подхода можно, например, безоговорочно утверждать, что если  $n$  асимптотически приближается к бесконечности, то алгоритм, характеризующийся показателем  $O(n)$ , проявляет себя лучше по сравнению с алгоритмом  $O(n^2)$ , тогда как единственное число, полученное с помощью эталонного тестирования, не может служить обоснованием подобного утверждения.

Система обозначений  $O()$  позволяет создать абстрактное представление, в котором не учитываются постоянные коэффициенты, благодаря чему она становится более простой в использовании, но менее точной, чем система обозначений  $T()$ . Например, в конечном итоге алгоритм  $O(n^2)$  всегда будет считаться худшим по сравнению с алгоритмом  $O(n)$ , но если бы эти два алгоритма характеризовались значениями  $T(n^2+1)$  и  $T(100n+1000)$ , то фактически алгоритм  $O(n^2)$  был бы лучше при  $n \leq 110$ .

Несмотря на этот недостаток, асимптотический анализ представляет собой инструментальное средство, наиболее широко используемое для анализа алгоритмов. Этот метод можно считать достаточно точным, поскольку в процессе анализа создается абстрактное представление и для точного количества операций (поскольку игнорируется постоянный коэффициент  $k$ ), и для точного содержимого входных данных (поскольку рассматривается исключительно их объем  $n$ ); благодаря этому анализ становится осуществимым с помощью математических методов. Система обозначений  $O()$  представляет собой хороший компромисс между точностью и простотой анализа.

### **Изначально сложные и недетерминированные полиномиальные задачи**

С помощью анализа алгоритмов и системы обозначений  $O()$  мы получаем возможность рассуждать об эффективности конкретного алгоритма. Однако эти методы не позволяют определить, может ли существовать лучший алгоритм для рассматриваемой задачи. В области **анализа сложности** исследуются задачи, а не алгоритмы. Первая широкая градация в этой области проводится между задачами, которые могут быть решены за время, измеряемое полиномиальным соотношением, и задачами, которые не могут быть решены за время, измеряемое полиномиальным соотношением, независимо от того, какой алгоритм для этого используется. Класс полиномиальных задач (которые могут быть решены за время  $O(n^k)$  для некоторого  $k$ ) обозначается как  $P$ . Эти задачи иногда называют “легкими”, поскольку данный класс содержит задачи, имеющие такую продолжительность прогона, как  $O(\log n)$  и  $O(n)$ . Но он содержит и задачи с затратами времени  $O(n^{1000})$ , поэтому определение “легкий” не следует понимать слишком буквально.

Еще одним важным классом является  $NP$  (сокращение от Nondeterministic Polynomial) — класс недетерминированных полиномиальных задач. Задача относится к этому классу, если существует алгоритм, позволяющий выдвинуть гипотезу о возможном решении, а затем проверить правильность этой гипотезы с помощью полиномиальных затрат времени. Идея такого подхода состоит в том, что если бы можно было воспользоваться сколь угодно большим количеством процессоров, чтобы проверить одновременно все гипотезы, или оказаться крайне удачливым и всегда с первого раза находить правильную гипотезу, то  $NP$ -трудные задачи стали бы  $P$ -трудными задачами. Одним из самых важных нерешенных вопросов в компьютерных науках является то, будет ли класс  $NP$  эквивалентным классу  $P$ , если нельзя воспользоваться бесконечным количеством процессоров или способностью находить правильную гипотезу с первого раза. Большинство специалистов в области компьютерных наук согласны с тем, что  $P \neq NP$ , иными словами, что задачи  $NP$  являются изначально трудными и для них не существует алгоритмов с полиномиальными затратами времени. Но это утверждение так и не было доказано.

Ученые, пытающиеся найти ответ на вопрос о том, эквивалентны ли классы  $P$  и  $NP$ , выделили подкласс класса  $NP$ , называемый **NP-полными** задачами. В этой формулировке слово “полный” означает “являющийся наиболее ярким представителем” и поэтому указывает на самые трудные задачи из класса  $NP$ . Было доказано, что либо все  $NP$ -полные задачи принадлежат к классу  $P$ , либо ни одна из них к нему не относится. Таким образом, данный класс представляет определенный теоретический интерес, но он важен также с точки зрения практики, поскольку известно, что

многие серьезные задачи являются NP-полными. В качестве примера можно указать задачу установления выполнимости: если дано высказывание пропозициональной логики, то есть ли такой вариант присваивания истинностных значений пропозициональным символам в высказывании, при котором оно становится истинным? Если не произойдет чудо и не совпадут друг с другом классы P и NP, то нельзя будет найти алгоритм, который позволяет решить все задачи установления выполнимости за полиномиальное время. Но исследователей в области искусственного интеллекта в большей степени интересует то, существуют ли алгоритмы, действующие достаточно эффективно при решении типичных задач, выбранных с помощью заранее заданного распределения; как было показано в главе 7, существуют алгоритмы наподобие WalkSAT, которые действуют вполне успешно при решении многих задач.

Класс  $\text{ко-}NP$  является комплементарным (дополнительным) по отношению к классу NP; это означает, что для каждой задачи принятия решения из класса NP существует соответствующая задача в классе ко-NP, на которую может быть дан положительный или отрицательный ответ, противоположный ответу на задачу класса NP. Известно, что P является подмножеством и NP, и ко-NP, кроме того, считается, что к классу ко-NP относятся некоторые задачи, не входящие в класс P. Такие задачи называются  $\text{ко-}NP\text{-полными}$  и являются самыми трудными задачами в классе ко-NP.

Класс #P (произносится как “шарп Р”, или “диэз Р”) представляет собой множество задач подсчета количества вариантов, соответствующих задачам принятия решения из класса NP. Задачи принятия решения имеют однозначный (положительный или отрицательный) ответ; примером задачи такого типа является следующая: “Существует ли решение для данной формулы 3-SAT?” Задачи подсчета количества вариантов имеют целочисленный ответ, например: “Сколько решений существует для данной формулы 3-SAT?” В некоторых случаях задача подсчета количества вариантов намного труднее по сравнению с соответствующей задачей принятия решения. Например, принятие решения о том, имеет ли двухдольный граф идеальное паросочетание, может быть выполнено за время  $O(VE)$  (где V — количество вершин; E — количество ребер графа), тогда как задача подсчета того, какое количество идеальных паросочетаний имеется в этом двухдольном графе, является #P-полной. Это означает, что она не менее трудна, чем любая задача из класса #P, и поэтому по меньшей мере столь же трудна, как и любая задача NP.

Исследователи определили также класс задач PSPACE; таковыми являются задачи, для которых требуется объем пространства, определяемый полиномиальной зависимостью, даже при их прогоне на недетерминированной машине. Считается, что PSPACE-трудные задачи решаются хуже по сравнению с NP-полными задачами, но не исключено, что в ходе дальнейших исследований может быть установлено, что класс NP эквивалентен классу PSPACE, и также то, что класс P эквивалентен классу NP.

## A.2. ВЕКТОРЫ, МАТРИЦЫ И ЛИНЕЙНАЯ АЛГЕБРА

В математике сформулировано определение  $\text{вектора}$  как элемента векторного пространства, но мы будем использовать более конкретное определение: вектор — это упорядоченная последовательность значений. Например, в двухмерном пространстве могут быть определены такие векторы, как  $\mathbf{x} = \langle 3, 4 \rangle$  и  $\mathbf{y} = \langle 0, 2 \rangle$ . В этом

приложении соблюдаются обычные соглашения об обозначении в нем векторов с помощью полужирных символов, хотя некоторые авторы отмечают имена векторов с помощью стрелок или знаков надчеркивания:  $\vec{x}$  или  $\vec{y}$ . Элементы вектора обозначаются с помощью подстрочных индексов:  $\mathbf{z} = \langle z_1, z_2, \dots, z_n \rangle$ .

Двумя фундаментальными операциями над векторами являются векторное сложение и скалярное умножение. Векторное сложение  $\mathbf{x} + \mathbf{y}$  — это поэлементная сумма:  $\mathbf{x} + \mathbf{y} = \langle 3+0, 4+2 \rangle = \langle 3, 6 \rangle$ , а скалярное умножение — это операция умножения каждого элемента на некоторую константу:  $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$ .

Длина вектора обозначается как  $|\mathbf{x}|$  и вычисляется путем извлечения квадратного корня из суммы квадратов элементов:  $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$ . Точечное произведение (называемое также скалярным произведением) двух векторов,  $\mathbf{x} \cdot \mathbf{y}$ , представляет собой сумму произведений соответствующих элементов; иными словами,

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i,$$

или в данном конкретном случае:  $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$ .

Векторы часто интерпретируются как направленные отрезки прямых (напоминающие стрелки) в  $n$ -мерном евклидовом пространстве. В таком случае операция сложения векторов эквивалентна совмещению конца одного вектора с началом другого, а точечное произведение  $\mathbf{x} \cdot \mathbf{y}$  эквивалентно выражению  $|\mathbf{x}| |\mathbf{y}| \cos \theta$ , где  $\theta$  — угол между  $\mathbf{x}$  и  $\mathbf{y}$ .

**Матрица** — это прямоугольный массив значений, упорядоченных по строкам и столбцам. Ниже показана матрица  $\mathbf{m}$  с размерами  $3 \times 4$ .

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \end{pmatrix}$$

Первый подстрочный индекс терма  $m_{i,j}$  определяет строку, а второй — столбец. В языках программирования терм  $m_{i,j}$  часто записывается как  $m[i, j]$  или  $m[i][j]$ .

Сумма двух матриц определяется путем сложения соответствующих элементов, поэтому  $(\mathbf{m} + \mathbf{n})_{i,j} = m_{i,j} + n_{i,j}$ . (Если матрицы  $\mathbf{m}$  и  $\mathbf{n}$  имеют разные размеры, то их сумма не определена.) Можно также определить операцию умножения матрицы на скаляр:  $(c\mathbf{m})_{i,j} = c m_{i,j}$ . Операция умножения матриц (получения произведения двух матриц) является более сложной. Произведение  $\mathbf{mn}$  определено, только если  $\mathbf{m}$  имеет размеры  $a \times b$ , а  $\mathbf{n}$  — размеры  $b \times c$  (т.е. вторая матрица имеет количество строк, совпадающее с количеством столбцов первой матрицы); результатом является матрица с размерами  $a \times c$ . Это означает, что операция умножения матриц не коммутативна — в общем случае  $\mathbf{mn} \neq \mathbf{nm}$ . Если матрицы имеют приемлемые размеры, то результат их умножения являются следующим:

$$(\mathbf{mn})_{i,k} = \sum_j m_{i,j} n_{j,k}$$

Единичная матрица  $\mathbf{I}$  имеет элементы  $I_{i,j}$ , равные 1, если  $i=j$ , и равные 0 в противном случае. Она обладает таким свойством, что  $\mathbf{mI} = \mathbf{m}$  при всех  $\mathbf{m}$ . Транспозиция матрицы  $\mathbf{m}$ , которая записывается как  $\mathbf{m}^T$ , образуется путем превращения строк в столбцы и наоборот, или, более формально, путем выполнения операции  $m_{i,j}^T = m_{j,i}$ .

Матрицы используются для решения систем линейных уравнений с помощью процедуры, называемой алгоритмом ~~удаления элементов Гаусса–Джордана~~, который характеризуется показателем  $O(n^3)$ . Рассмотрим следующее множество уравнений, для которого можно найти решение в терминах  $x$ ,  $y$  и  $z$ :

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

Эту систему уравнений можно представить в виде следующей матрицы:

$$\left( \begin{array}{ccc|c} x & y & z & c \\ 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array} \right)$$

Здесь строка  $x$   $y$   $z$   $c$  не входит в состав матрицы; она служит лишь в качестве напоминания, к чему относится каждый столбец. Известно, что если обе стороны уравнения будут умножены на константу или если это уравнение будет сложено с другим уравнением, то полученное уравнение останется действительным. Метод удаления элементов Гаусса–Джордана действует по принципу повторного выполнения подобных операций таким образом, чтобы вначале была удалена первая переменная ( $x$ ) из всех уравнений, кроме первого, а затем эти действия продолжались в форме удаления  $i$ -й переменной из всех уравнений, кроме  $i$ -го, для всех  $i$ . Удалим  $x$  из второго уравнения, умножив первое уравнение на  $3/2$  и сложив его со вторым. В результате будет получена следующая матрица:

$$\left( \begin{array}{ccc|c} x & y & z & c \\ 2 & 1 & -1 & 8 \\ 0 & .5 & .5 & 1 \\ -2 & 1 & 2 & -3 \end{array} \right)$$

Продолжая аналогичные действия, удалим  $x$ ,  $y$  и  $z$  и получим следующее:

$$\left( \begin{array}{ccc|c} x & y & z & c \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array} \right)$$

Таким образом, решением является  $x=2$ ,  $y=3$ ,  $z=-1$  (проверьте это решение!).

### A.3. РАСПРЕДЕЛЕНИЯ ВЕРОЯТНОСТЕЙ

Вероятность — это мера, заданная на множестве событий, которая удовлетворяет трем приведенным ниже аксиомам.

1. Мера каждого события находится в пределах от 0 до 1. Это утверждение записывается как  $0 \leq P(E=e_i) \leq 1$ , где  $E$  — случайная переменная, представляющая событие;  $e_i$  — возможные значения  $E$ . Вообще говоря, случайные переменные обозначаются прописными буквами, а их значения — строчными.
2. Мера всего множества равна 1; это означает, что

$$\sum_{i=1}^n P(E=e_i) = 1.$$

3. Вероятность объединения несовместимых событий равна сумме вероятностей отдельных событий; это означает, что  $P(E=e_1 \vee E=e_2) = P(E=e_1) + P(E=e_2)$ , где  $e_1$  и  $e_2$  являются несовместимыми.

**Вероятностная модель** состоит из пространства выборок взаимоисключающих возможных результатов, наряду с вероятностной мерой для каждого результата. Например, в модели погоды на завтра результатами могут быть *sunny* (солнце), *cloudy* (облачность), *rainy* (дождь) и *snowy* (снег). Подмножество этих результатов представляет собой событие. Например, событие выпадения осадков — это подмножество  $\{rainy, snowy\}$ .

Для обозначения вектора значений  $\langle P(E=e_1), \dots, P(E=e_n) \rangle$  используется термин  $\mathbf{P}(E)$ . Кроме того,  $P(e_i)$  применяется как сокращенное обозначение для  $P(E=e_i)$ , а  $\sum_e P(e)$

служит для обозначения

$$\sum_{i=1}^n P(E=e_i).$$

Условная вероятность  $P(B|A)$  определяется как  $P(B \cap A) / P(A)$ . Переменные  $A$  и  $B$  являются условно независимыми, если  $P(B|A) = P(B)$  (или, равным образом, если  $P(A|B) = P(A)$ ). Непрерывные переменные имеют бесконечное количество значений, и если распределение вероятностей этих значений не характеризуется наличием пиковых значений в отдельных точках, то вероятность любого отдельно взятого значения равна 0. Поэтому определим **функцию плотности вероятностей**, которая также обозначается как  $P(X)$ , но имеет немного иной смысл по сравнению с дискретной функцией вероятностей  $P(A)$ . Функция плотности вероятностей  $P(X=c)$  определяется как отношение вероятности того, что  $X$  попадает в интервал вокруг  $c$ , к ширине этого интервала, измеряемая в пределе приближения ширины интервала к нулю:

$$P(X=c) = \lim_{dx \rightarrow 0} P(c \leq X \leq c + dx) / dx$$

Функция плотности должна быть неотрицательной при всех  $x$  и соответствовать следующему требованию:

$$\int_{-\infty}^{\infty} P(X) dx = 1$$

Может быть также определена **кумулятивная функция плотности вероятностей**  $F(X)$ , которая представляет собой вероятность того, что некоторая случайная переменная меньше  $x$ :

$$F(X) = \int_{-\infty}^x P(X) dx$$

Следует отметить, что функция плотности вероятностей измеряется в определенных единицах, а дискретная функция вероятностей является безразмерной. Например, если переменная  $X$  измеряется в секундах, то плотность измеряется в Гц (т.е.  $1/c$ ). Если  $x$  — точка в трехмерном пространстве, измеряемом в метрах, то плотность измеряется в  $1/m^3$ .

Одним из наиболее важных распределений вероятностей является **гауссово распределение**, известное также под названием **нормального распределения**. Гауссово распределение со средним  $\mu$  и среднеквадратичным отклонением  $\sigma$  (и поэтому с дисперсией  $\sigma^2$ ) определяется следующей формулой:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

где  $x$  — непрерывная переменная, изменяющаяся в пределах от  $-\infty$  до  $+\infty$ . Если среднее  $\mu=0$  и дисперсия  $\sigma^2=1$ , то имеет место частный случай нормального распределения, называемый **стандартным нормальным распределением**. Если распределение задано на векторе  $\mathbf{x}$  в пространстве с  $d$  измерениями, то оно представляет собой **многомерное гауссово распределение**:

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2} (\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)}$$

где  $\mu$  — вектор средних;  $\Sigma$  — **матрица ковариации** этого распределения.

При наличии одного измерения можно также определить функцию **кумулятивного распределения**  $F(x)$  как вероятность того, что случайная переменная будет меньше чем  $x$ . Для стандартного нормального распределения эта функция задается следующим образом:

$$F(x) = \int_{-\infty}^x P(x) dx = \frac{1}{2} \left( 1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right) \right)$$

где  $\operatorname{erf}(x)$  — так называемая **функция ошибок**, не имеющая представления в замкнутой форме.

В **центральной предельной теореме** утверждается, что среднее  $n$  случайных переменных приближается к нормальному распределению по мере того, как  $n$  стремится к бесконечности. Такому свойству подчиняется почти любая коллекция случайных переменных, при том условии, что значение дисперсии любого конечного подмножества переменных не доминирует над значениями дисперсии других конечных подмножеств.

## БИБЛИОГРАФИЧЕСКИЕ И ИСТОРИЧЕСКИЕ ЗАМЕТКИ

Система обозначений  $O()$ , которая в наши дни широко используется в компьютерных науках, была впервые определена в контексте теории чисел немецким математиком П.Г. Н. Бахманом [57]. Понятие NP-полноты было предложено Куком [289], а современный метод определения способа сведения одной проблемы к другой предложен Карпом [772]. И Кук, и Карп получили за свою работу премию Тьюринга — высшую награду в компьютерных науках.

В число классических работ по анализу и проектированию алгоритмов по праву входят [8] и [809]; более современные достижения отражены в [295] и [1489]. Эти

книги в основном посвящены проектированию и анализу алгоритмов решения разрешимых задач. Сведения в области теории NP-полноты и других форм неразрешимости приведены в книгах Гэри и Джонсона [520], а также Пападимитриу [1168]. В своей книге Гэри и Джонсон не только изложили теоретические основы, но и привели примеры, наглядно показывающие, по каким причинам специалисты в области компьютерных наук считают бесспорным утверждение, что линии водораздела между разрешимыми и неразрешимыми задачами проходит по границе между полиномиальной и экспоненциальной временной сложностью. Кроме того, эти ученые представили объемистый каталог задач, известных как NP-полные или неразрешимые по каким-то другим критериям.

К числу хороших учебников по теории вероятностей относятся [122], [254], [463] и [1310].

# Б ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКАХ И АЛГОРИТМАХ, ИСПОЛЬЗУЕМЫХ В КНИГЕ

## Б.1. ОПРЕДЕЛЕНИЕ ЯЗЫКОВ С ПОМОЩЬЮ ФОРМЫ БЭКУСА–НАУРА

В настоящей книге определено несколько языков, включая языки пропозициональной логики (с. 294), логики первого порядка (с. 349) и подмножество английского языка (с. 1066). Формальный язык определяется как множество строк, в котором каждая строка представляет собой последовательность символов. Все языки, которые были необходимы нам в этой книге, состоят из бесконечного множества строк, поэтому нужен краткий способ, позволяющий охарактеризовать это множество. Для этого служит **грамматика**. Авторы приняли в качестве способа оформления грамматик формальную систему, называемую **формой Бэкуса–Наура** (*Backus-Naur form* — BNF). Грамматика BNF состоит из четырех компонентов, перечисленных ниже.

- Множество **терминальных символов**. Это — символы или слова, из которых состоят строки языка. В качестве таких символов могут использоваться буквы (**A**, **B**, **C**, ...) или слова (**a**, **aardvark**, **abacus**, ...).
- Множество **нетерминальных символов**, которые обозначают компоненты предложений языка. Например, нетерминальный символ *NounPhrase* (именное словосочетание) в английском языке обозначает бесконечное множество строк, которое включает “you” и “the big slobbery dog”.
- **Начальный символ**, который представляет собой нетерминальный символ, обозначающий законченные строки языка. В описанной выше грамматике

английского языка таковым является символ *Sentence*; в грамматике арифметических выражений может быть задан символ *Expr*.

- Множество **правил подстановки** в форме  $LHS \rightarrow RHS$ , где *LHS* — нетерминальный символ; *RHS* — последовательность, в которую входят от нуля и больше символов (терминальных или нетерминальных).

Правило подстановки в приведенной ниже форме означает, что при наличии двух строк, обозначенных как *NounPhrase* (именное словосочетание) и *VerbPhrase* (глагольное словосочетание), их можно соединить друг с другом и обозначить результат как *Sentence*:

*Sentence*  $\rightarrow$  *NounPhrase* *VerbPhrase*

Несколько правил с одинаковой левой частью могут быть заданы как одно правило с той же левой частью и со всеми правыми частями этих правил, разделенными символом | . Ниже приведена грамматика BNF для простых арифметических выражений.

<i>Expr</i>	$\rightarrow$	<i>Expr Operator Expr</i>		<i>( Expr )</i>		<i>Number</i>														
<i>Number</i>	$\rightarrow$	<i>Digit</i>		<i>Number Digit</i>																
<i>Digit</i>	$\rightarrow$	0		1		2		3		4		5		6		7		8		9
<i>Operator</i>	$\rightarrow$	+		-		$\div$		$\times$												

Более подробные сведения о языках и грамматиках приведены в главе 22. Следует отметить, что в других книгах в системе обозначений BNF могут использоваться немного другие правила, например, нетерминальные символы обозначаются как  $\langle Digit \rangle$  вместо *Digit*, терминальные символы — 'word', а не **word**, а в правилах используются символ : := вместо  $\rightarrow$ .

## Б.2. ОПИСАНИЕ АЛГОРИТМОВ С ПОМОЩЬЮ ПСЕВДОКОДА

В настоящей книге подробно определено свыше 80 алгоритмов. Авторы решили не останавливаться на каком-то одном языке программирования (и рисовать тем, что читатели, не знакомые с этим языком, не смогут воспользоваться приведенными здесь алгоритмами) и вместо этого представить алгоритмы на псевдокоде. Основные конструкции этого псевдокода должны быть знакомы пользователям таких языков, как Java, C++ и Lisp. В некоторых местах для описания вычислений используются математические формулы или текст на естественном языке, поскольку в противном случае пришлось бы применять более громоздкие конструкции. Необходимо также отметить, что в алгоритмах используются приведенные ниже соглашения.

- **Статические переменные.** Ключевое слово *static* используется для указания на то, что переменной присваивается начальное значение при первом вызове некоторой функции, после чего это значение (или значение, присвоенное переменной с помощью выполненных в дальнейшем операторов присваивания) сохраняется при всех последующих вызовах функции. Таким образом, статические переменные напоминают глобальные переменные в том, что они сохраняют свое значение после каждого вызова содержащей их функции, но остаются доступными только в этой функции. В программах агентов, приведенных в данной книге, статические переменные используются в качестве

“оперативной памяти”. Программы со статическими переменными могут быть реализованы в объектно-ориентированных языках, таких как Java и Smalltalk, в виде “объектов”. В функциональных языках они могут быть реализованы с помощью функциональных замыканий в той среде, в которой определены требуемые переменные.

- **Функции как значения.** Имена функций и процедур начинаются с прописных букв, а имена переменных состоят из строчных букв и выделяются курсивом. Поэтому чаще всего вызов функции выглядит наподобие  $F_n(x)$ . Однако допускается, чтобы значением переменной была функция; например, если значением переменной  $f$  является функция вычисления квадратного корня, то выражение  $f(9)$  возвращает 3.
- **Начальный индекс массива, равный 1.** Если не указано иное, то первым индексом массива является 1, как в обычной системе математических обозначений, а не 0, как в языках Java и C.
- **Значимость отступов.** По аналогии с языком Python и в отличие от языков Java и C++ (в которых используются фигурные скобки) или языков Pascal и Visual Basic (в которых используется оператор `end`) для обозначения области действия цикла или условного выражения применяются отступы.

### Б.3. ОПЕРАТИВНАЯ ПОМОЩЬ

---

Большинство алгоритмов, приведенных в данной книге, реализовано на нескольких языках программирования и представлено в нашем оперативном репозитории кода по следующему адресу:

☞ [aima.cs.berkeley.edu](mailto:aima.cs.berkeley.edu)

Если у вас есть какие-либо комментарии, если вы обнаружили какие-либо неточности или хотите внести предложения по усовершенствованию данной книги, мы всегда будем рады вас выслушать. Посетите указанный выше Web-узел, чтобы ознакомиться с инструкциями по подключению к дискуссионной группе по интересам или отправьте по электронной почте сообщение по следующему адресу:

☞ [aima@cs.berkeley.edu](mailto:aima@cs.berkeley.edu)

# ОПИСАНИЕ СОПРО- ВОЖДАЮЩЕГО WEB-УЗЛА

Авторы книги “Искусственный интеллект. Современный подход. Второе издание” подготовили дополнительные материалы к книге, которые находятся на сопровождающем Web-узле по адресу [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu). Первая страница этого узла показана на рис. 1.

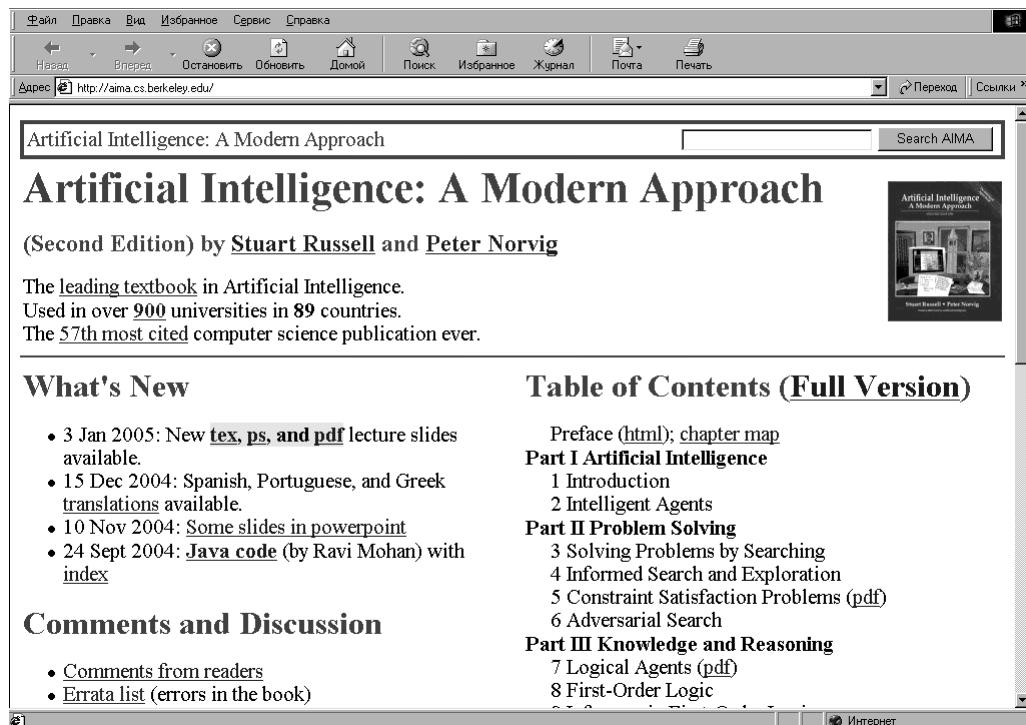


Рис. 1. Первая страница Web-узла [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu)

Материалы, представленные на этом узле, постоянно обновляются и ко времени выхода из печати настоящего издания включают следующее:

- псевдокод алгоритмов, описанных в книге, в формате PDF и PS;
- реализаций представленных алгоритмов на нескольких языках программирования (оперативный репозитарий кода);

- данные, предназначенные для проверки программ из репозитария кода;
- аплеты Java и сценарии JavaScript, предназначенные для демонстрации в оперативном режиме;
- список более чем 900 учебных заведений, в которых используется данная книга, сопровождающийся многочисленными ссылками на материалы курсов, доступные в оперативном режиме;
- аннотированный список более чем 900 ссылок на Web-узлы с полезными сведениями по искусственному интеллекту;
- списки дополнительных материалов и ссылок, относящихся к каждой главе;
- инструкции с описанием того, как присоединяться к дискуссионной группе, посвященной данной книге;
- инструкции с описанием того, как обратиться к авторам, чтобы передать им свои вопросы или комментарии;
- инструкции с описанием того, как сообщить об ошибках, обнаруженных в книге;
- копии рисунков из оригинала книги, а также слайды и другие материалы для преподавателей.

# ЛИТЕРАТУРА

1. Aarup M., Arentoft M. M., Parrod Y., Stader J., and Stokes I. (1994) OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox M. and Zweben M. (Eds.) *Knowledge Based Scheduling*. Morgan Kaufmann, San Mateo, California.
2. Abramson B. and Yung M. (1989) Divide and conquer under global constraints: A solution to the N-queens problem. *Journal of Parallel and Distributed Computing*, 6(3), p. 649–662.
3. Ackley D. H. and Littman M. L. (1991) Interactions between learning and evolution. In Langton C., Taylor C., Farmer J. D., and Ramussen S. (Eds.) *Artificial Life II*, p. 487–509. Addison-Wesley, Redwood City, California.
4. Adelson-Velsky G. M., Arlazarov V. L., Bitman A. R., Zhivotovsky A. A., and Uskov A. V. (1970) Programming a computer to play chess. *Russian Mathematical Surveys*, 25, p. 221–262.
5. Adelson-Velsky G. M., Arlazarov V. L., and Donskoy M. V. (1975) Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, 6(4), p. 361–371.
6. Agmon S. (1954) The relaxation method for linear inequalities. *Canadian J. Math.*, 6(3), p. 382–392.
7. Agre P. E. and Chapman D. (1987) Pengi: an implementation of a theory of activity. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, p. 268–272, Milan. Morgan Kaufmann.
8. Aho A. V., Hopcroft J., and Ullman J. D. (1974) *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts.
9. Aho A. V. and Ullman J. D. (1972) *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, Upper Saddle River, New Jersey.
10. Ait-Kaci H. and Podelski A. (1993) Towards a meaning of LIFE. *Journal of Logic Programming*, 16(3–4), p. 195–234.
11. Aizerman M., Braverman E., and Rozonoer L. (1964) Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, p. 821–837.
12. Albus J. S. (1975) A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 97, p. 270–277.
13. Aldous D. and Vazirani U. (1994) “Go with the winners” algorithms. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, p. 492–501, Santa Fe, New Mexico. IEEE Computer Society Press.
14. Allais M. (1953) Le comportment de l’homme rationnel devant la risque: critique des postulats et axiomes de l’école Américaine. *Econometrica*, 21, p. 503–546.
15. Allen J. F. (1983) Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery*, 26(11), p. 832–843.
16. Allen J. F. (1984) Towards a general theory of action and time. *Artificial Intelligence*, 23, p. 123–154.
17. Allen J. F. (1991) Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6, p. 341–355.
18. Allen J. F. (1995) *Natural Language Understanding*. Benjamin/Cummings, Redwood City; California.
19. Allen J. F., Hendler J., and Tate A. (Eds.) (1990) *Readings in Planning*. Morgan Kaufmann, San Mateo, California.

20. Almuallim H. and Dietterich T. (1991) Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Vol. 2, p. 547–552, Anaheim, California. AAAI Press.
21. ALPAC (1966) Language and machines: Computers in translation and linguistics. Tech. rep. 1416, The Automatic Language Processing Advisory Committee of the National Academy of Sciences, Washington, DC.
22. Alshawi H. (Ed.) (1992) *The Core Language Engine*. MIT Press, Cambridge, Massachusetts.
23. Alterman R. (1988) Adaptive planning. *Cognitive Science*, 12, p. 393–422.
24. Amarel S. (1968) On representations of problems of reasoning about actions. In Michie D. (Ed.), *Machine Intelligence 3*, Vol. 3, p. 131–171. Elsevier/North-Holland, Amsterdam, London, New York.
25. Ambros-Ingerson J. and Steel S. (1988) Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, p. 735–740, St. Paul, Minnesota. Morgan Kaufmann.
26. Amit D., Gutfreund H., and Sompolinsky H. (1985) Spin-glass models of neural networks. *Physical Review, A* 32, p. 1007–1018.
27. Andersen S. K., Olesen K. G., Jensen F. V., and Jensen F. (1989) HUGIN — a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 2, p. 1080–1085, Detroit. Morgan Kaufmann.
28. Anderson A. R. (Ed.) (1964) *Minds and Machines*. Prentice-Hall, Upper Saddle River, New Jersey.
29. Anderson J. A. and Rosenfeld E. (Eds.) (1988) *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, Massachusetts.
30. Anderson J. R. (1980) *Cognitive Psychology and Its Implications*. W. H. Freeman, New York.
31. Anderson J. R. (1983) *The Architecture of Cognition*. Harvard University Press, Cambridge, Massachusetts.
32. Andre D. and Russell S. J. (2002) State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, p. 119–125, Edmonton, Alberta. AAAI Press.
33. Anshelevich V. A. (2000) The game of Hex: An automatic theorem proving approach to game programming. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, p. 189–194, Austin, Texas. AAAI Press.
34. Anthony M. and Bartlett P. (1999) *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, Cambridge, UK.
35. Appel K. and Haken W. (1977) Every planar map is four colorable: Part I: Discharging. *Illinois J. Math.*, 21, p. 429–490.
36. Apt K. R. (1999) The essence of constraint propagation. *Theoretical Computer Science*, 221(1–2), p. 179–210.
37. Apté, C., Damerau F., and Weiss S. (1994) Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12, p. 233–251.
38. Arkin R. (1998) *Behavior-Based Robotics*. MIT Press, Boston, MA.
39. Armstrong D. M. (1968) *A Materialist Theory of the Mind*. Routledge and Kegan Paul, London.
40. Arnauld A. (1662) *La logique, ou l'art de penser*. Chez Charles Savreux, au pied de la Tour de Nostre Dame, Paris.
41. Arora S. (1998) Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the Association for Computing Machinery*, 45(5), p. 753–782.
42. Ashby W. R. (1940) Adaptiveness and equilibrium. *Journal of Mental Science*, 86, p. 478–483.
43. Ashby W. R. (1948) Design for a brain. *Electronic Engineering*, December, p. 379–383.

44. Ashby W. R. (1952) *Design for a Brain*. Wiley, New York.
45. Asimov I. (1942) Runaround. *Astounding Science Fiction, March*.
46. Asimov I. (1950) *I, Robot*. Doubleday, Garden City, New York.
47. Astrom K. J. (1965) Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10, p. 174–205.
48. Audi R. (Ed.) (1999) *The Cambridge Dictionary of Philosophy*. Cambridge University Press, Cambridge, UK.
49. Austin J. L. (1962) *How To Do Things with Words*. Harvard University Press, Cambridge, Massachusetts.
50. Axelrod R. (1985) *The Evolution of Cooperation*. Basic Books, New York.
51. Bacchus F. (1990) *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, Massachusetts.
52. Bacchus F. and Grove A. (1995) Graphical models for preference and utility. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, p. 3–10, Montreal, Canada. Morgan Kaufmann.
53. Bacchus F. and Grove A. (1996) Utility independence in a qualitative decision theory. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning*, p. 542–552, San Mateo, California. Morgan Kaufmann.
54. Bacchus F., Grove A., Halpern J. Y., and Koller D. (1992) From statistics to beliefs. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, p. 602–608, San Jose. AAAI Press.
55. Bacchus F. and van Beek P. (1998) On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 311–318, Madison, Wisconsin. AAAI Press.
56. Bacchus F. and van Run P. (1995) Dynamic variable ordering in CSPs. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, p. 258–275, Cassis, France. Springer-Verlag.
57. Bachmann P. G. H. (1894) *Die analytische Zahlentheorie*. B. G. Teubner, Leipzig.
58. Backus J. W. (1996) Transcript of question and answer session. In Wexelblat R. L. (Ed.), *History of Programming Languages*, p. 162. Academic Press, New York.
59. Baeza-Yates R. and Ribeiro-Neto B. (1999) *Modern Information Retrieval*. Addison Wesley Longman, Reading, Massachusetts.
60. Bajcsy R. and Lieberman L. (1976) Texture gradient as a depth cue. *Computer Graphics and Image Processing*, 5(1), p. 52–67.
61. Baker C. L. (1989) *English Syntax*. MIT Press, Cambridge, Massachusetts.
62. Baker J. (1975) The Dragon system — an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23, p. 24–29.
63. Baker J. (1979) Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, p. 547–550, Cambridge, Massachusetts. MIT Press.
64. Baldwin J. M. (1896) A new factor in evolution. *American Naturalist*, 30, p. 441–451. Продолжение на с. 536–553.
65. Ballard B. W. (1983) The \*-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3), p. 327–350.
66. Baluja S. (1997) Genetic algorithms and explicit search statistics. In Mozer M. C., Jordan M. I., and Petsche T. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 9, p. 319–325. MIT Press, Cambridge, Massachusetts.
67. Bancilhon F., Maier D., Sagiv Y., and Ullman J. D. (1986) Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth ACM Symposium on Principles of Database Systems*, p. 1–16, New York. ACM Press.
68. Bar-Hillel Y. (1954) Indexical expressions. *Mind*, 63, p. 359–379.

69. Bar-Hillel Y. (1960) The present status of automatic translation of languages. In Alt F. L. (Ed.), *Advances in Computers*, Vol. 1, p. 91–163. Academic Press, New York.
70. Bar-Shalom Y. (Ed.) (1992) *Multitarget-multisensor tracking: Advanced applications*. Artech House, Norwood, Massachusetts.
71. Bar-Shalom Y. and Fortmann T. E. (1988) *Tracking and Data Association*. Academic Press, New York.
72. Barrett A. and Weld D. S. (1994) Task-decomposition via plan parsing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1117–1122, Seattle. AAAI Press.
73. Bartak R. (2001) Theory and practice of constraint propagation. In *Proceedings of the Third Workshop on Constraint Programming for Decision and Control (CPDC-01)*, p. 7–14, Gliwice, Poland.
74. Barto A. G., Bradtke S. J., and Singh S. P. (1995) Learning to act using real-time dynamic programming. *Artificial Intelligence*, 73(1), p. 81–138.
75. Barto A. G., Sutton R. S., and Anderson C. W. (1983) Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13, p. 834–846.
76. Barto A. G., Sutton R. S., and Brouwer P. S. (1981) Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 40(3), p. 201–211.
77. Barton G. E., Berwick R. C., and Ristad E. S. (1987) *Computational Complexity and Natural Language*. MIT Press, Cambridge, Massachusetts.
78. Barwise J. and Etchemendy J. (1993) *The Language of First-Order Logic: Including the Macintosh Program Tarski's World 4.0* (Third Revised and Expanded edition) Center for the Study of Language and Information (CSLI), Stanford, California.
79. Bateman J. A. (1997) Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering*, 3(1), p. 15–55.
80. Bateman J. A., Kasper R. T., Moore J. D., and Whitney R. A. (1989) A general organization of knowledge for natural language processing: The penman upper model. Tech. rep., Information Sciences Institute, Marina del Rey, CA.
81. Baum E., Boneh D., and Garrett C. (1995) On genetic algorithms. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory (COLT-92)*, p. 230–239, Santa Cruz, California. ACM Press.
82. Baum E. and Haussler D. (1989) What size net gives valid generalization? *Neural Computation*, 1(1), p. 151–160.
83. Baum E. and Smith W. D. (1997) A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2), p. 195–242.
84. Baum E. and Wilczek F. (1988) Supervised learning of probability distributions by neural networks. In Anderson D. Z. (Ed.), *Neural Information Processing Systems*, p. 52–61. American Institute of Physics, New York.
85. Baum L. E. and Petrie T. (1966) Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 41.
86. Baxter J. and Bartlett P. (2000) Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning*, p. 41–48, Stanford, California. Morgan Kaufmann.
87. Bayardo R. J. and Schrag R. C. (1997) Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, p. 203–208, Providence, Rhode Island. AAAI Press.
88. Bayes T. (1763) An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53, p. 370–418.
89. Beal D. F. (1980) An analysis of minimax. In Clarke M. R. B. (Ed.), *Advances in Computer Chess 2*, p. 103–109. Edinburgh University Press, Edinburgh, Scotland.
90. Beal D. F. (1990) A generalised quiescence search algorithm. *Artificial Intelligence*, 43(1), p. 85–98.

91. Beckert B. and Posegga J. (1995) Leantap: Lean, tableau-based deduction. *Journal of Automated Reasoning*, 15(3), p. 339–358.
92. Beeri C., Fagin R., Maier D., and Yannakakis M. (1983) On the desirability of acyclic database schemes. *Journal of the Association for Computing Machinery*, 30(3), p. 479–513.
93. Bell C. and Tate A. (1985) Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Alvey IKBS SIG Workshop*, Sunningdale, Oxfordshire, UK. Institution of Electrical Engineers.
94. Bell J. L. and Machover M. (1977) *A Course in Mathematical Logic*. Elsevier/North-Holland, Amsterdam, London, New York.
95. Bellman R. E. (1978) *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, San Francisco.
96. Bellman R. E. and Dreyfus S. E. (1962) *Applied Dynamic Programming*. Princeton University Press, Princeton, New Jersey.
97. Bellman R. E. (1957) *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.
98. Belongie S., Malik J., and Puzicha J. (2002) Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(4), p. 509–522.
99. Bender E. A. (1996) *Mathematical methods in artificial intelligence*. IEEE Computer Society Press, Los Alamitos, California.
100. Bentham J. (1823) *Principles of Morals and Legislation*. Oxford University Press, Oxford, UK. Original work published in 1789.
101. Berger J. O. (1985) *Statistical Decision Theory and bayesian Analysis*. Springer Verlag, Berlin.
102. Berlekamp E. R., Conway J. H., and Guy R. K. (1982) *Winning Ways, For Your Mathematical Plays*. Academic Press, New York.
103. Berleur J. and Brunnstein K. (2001) *Ethics of Computing: Codes, Spaces for Discussion and Law*. Chapman and Hall, London.
104. Berliner H. J. (1977) BKG — a program that plays backgammon. Tech. rep., Computer Science Department, Carnegie-Mellon University, Pittsburgh.
105. Berliner H. J. (1979) The B\* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12(1), p. 23–40.
106. Berliner H. J. (1980a) Backgammon computer program beats world champion. *Artificial Intelligence*, 14, p. 205–220.
107. Berliner H. J. (1980b) Computer backgammon. *Scientific American*, 249(6), p. 64–72.
108. Berliner H. J. and Ebeling C. (1989) Pattern knowledge and search: The SUPREM architecture. *Artificial Intelligence*, 38(2), p. 161–198.
109. Bernardo J. M. and Smith A. F. M. (1994) *Bayesian Theory*. Wiley, New York.
110. Berners-Lee T., Hendler J., and Lassila O. (2001) The semantic web. *Scientific American*, 284(5), p. 34–43.
111. Bernoulli D. (1738) Specimen theoriae novae de mensura sortis. *Proceedings of the St. Petersburg Imperial Academy of Sciences*, 5, p. 175–192.
112. Bernstein A. and Roberts M. (1958) Computer vs. chess player. *Scientific American*, 198(6), p. 96–105.
113. Bernstein A., Roberts M., Arbuckle T., and Belsky M. S. (1958) A chess playing program for the IBM 704. In *Proceedings of the 1958 Western Joint Computer Conference*, p. 157–159, Los Angeles. American Institute of Electrical Engineers.
114. Bernstein P. L. (1996) *Against the Odds: The Remarkable Story of Risk*. Wiley, New York.
115. Berrou C., Glavieux A., and Thitimajshima P. (1993) Near Shannon limit error control-correcting coding and decoding: Turbo-codes. 1. In *Proc. IEEE International Conference on Communications*, p. 1064–1070, Geneva, Switzerland. IEEE.

116. Berry D. A. and Fristedt B. (1985) *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London.
117. Bertele U. and Brioschi F. (1972) *Nonserial dynamic programming*. Academic Press, New York.
118. Bertoli P., Cimatti A., and Roveri M. (2001a) Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, p. 467–472, Seattle. Morgan Kaufmann.
119. Bertoli P., Cimatti A., Roveri M., and Traverso P. (2001b) Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, p. 473–478, Seattle. Morgan Kaufmann.
120. Bertsekas D. (1987) *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Upper Saddle River, New Jersey.
121. Bertsekas D. and Tsitsiklis J. N. (1996) *Neuro-dynamic programming*. Athena Scientific, Belmont, Massachusetts.
122. Bertsekas D. and Tsitsiklis J. N. (2002) *Introduction to Probability*. Athena Scientific, Belmont, Massachusetts.
123. Bibel W. (1981) On matrices with connections. *Journal of the Association for Computing Machinery*, 28(4), p. 633–645.
124. Bibel W. (1993) *Deduction: Automated Logic*. Academic Press, London.
125. Biggs N. L., Lloyd E. K., and Wilson R. J. (1986) *Graph Theory 1736–1936*. Oxford University Press, Oxford, UK.
126. Binder J., Koller D., Russell S. J., and Kanazawa K. (1997a) Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29, p. 213–244.
127. Binder J., Murphy K., and Russell S. J. (1997b) Space-efficient inference in dynamic probabilistic networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, p. 1292–1296, Nagoya, Japan. Morgan Kaufmann.
128. Binford T. O. (1971) Visual perception by computer. Invited paper presented at the IEEE Systems Science and Cybernetics Conference, Miami.
129. Binmore K. (1982) *Essays on Foundations of Game Theory*. Pitman, London.
130. Birnbaum L. and Selfridge M. (1981) Conceptual analysis of natural language. In Schank R. and Riesbeck C. (Eds.), *Inside Computer Understanding*. Lawrence Erlbaum, Potomac, Maryland.
131. Biro J. I. and Shaham R. W. (Eds.) (1982) *Mind, Brain and Function: Essays in the Philosophy of Mind*. University of Oklahoma Press, Norman, Oklahoma.
132. Birtwistle G., Dahl O.-J., Myrhaug B., and Nygaard K. (1973) *Simula Begin*. Studentlitteratur (Lund) and Auerbach, New York.
133. Bishop C. M. (1995) *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.
134. Bistarelli S., Montanari U., and Rossi F. (1997) Semiring-based constraint satisfaction and optimization. *Journal of the Association for Computing Machinery*, 44(2), p. 201–236.
135. Bitner J. R. and Reingold E. M. (1975) Backtrack programming techniques. *Communications of the Association for Computing Machinery*, 18(11), p. 651–656.
136. Blei D. M., Ng A. Y., and Jordan M. I. (2001) Latent Dirichlet Allocation. In *Neural Information Processing Systems*, Vol. 14, Cambridge, Massachusetts. MIT Press.
137. Blinder A. S. (1983) Issues in the coordination of monetary and fiscal policies. In *Monetary Policy Issues in the 1980s*. Federal Reserve Bank, Kansas City, Missouri.
138. Block N. (Ed.) (1980) *Readings in Philosophy of Psychology*, Vol. 1. Harvard University Press, Cambridge, Massachusetts.

139. Blum A. L. and Furst M. (1995) Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, p. 1636–1642, Montreal. Morgan Kaufmann.
140. Blum A. L. and Furst M. (1997) Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2), p. 281–300.
141. Blumer A., Ehrenfeucht A., Haussler D., and Warmuth M. (1989) Learnability and the Vapnik–Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4), p. 929–965.
142. Bobrow D. G. (1967) Natural language input for a computer problem solving system. In Minsky M. L. (Ed.), *Semantic Information Processing*, p. 133–215. MIT Press, Cambridge, Massachusetts.
143. Bobrow D. G., Kaplan R., Kay M., Norman D. A., Thompson H., and Winograd T. (1977) GUS, a frame driven dialog system. *Artificial Intelligence*, 8, p. 155–173.
144. Bobrow D. G. and Raphael B. (1974) New programming languages for artificial intelligence research. *Computing Surveys*, 6(3), p. 153–174.
145. Boden M. A. (1977) *Artificial Intelligence and Natural Man*. Basic Books, New York.
146. Boden M. A. (Ed.) (1990) *The Philosophy of Artificial Intelligence*. Oxford University Press, Oxford, UK.
147. Bonet B. and Geffner H. (1999) Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, p. 360–372, Durham, UK. Springer-Verlag.
148. Bonet B. and Geffner H. (2000) Planning with incomplete information as heuristic search in belief space. In Chien S., Kambhampati S., and Knoblock C. A. (Eds.), *International Conference on Artificial Intelligence Planning and Scheduling*, p. 52–61, Menlo Park, California. AAAI Press.
149. Boole G. (1847) *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*. Macmillan, Barclay, and Macmillan, Cambridge.
150. Boolos G. S. and Jeffrey R. C. (1989) *Computability and Logic* (3rd edition). Cambridge University Press, Cambridge, UK.
151. Booth T. L. (1969) Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, p. 74–81, Waterloo, Ontario. IEEE.
152. Borel E. (1921) La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 173, p. 1304–1308.
153. Borenstein J., Everett B., and Feng L. (1996) *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley, MA.
154. Borenstein J. and Koren Y. (1991) The vector field histogram — fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3), p. 278–288.
155. Borgida A., Brachman R. J., McGuinness D. L., and Alperin Resnick L. (1989) CLASSIC: A structural data model for objects. *SIGMOD Record*, 18(2), p. 58–67.
156. Boser B. E., Guyon I. M., and Vapnik V. N. (1992) A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*, Pittsburgh, Pennsylvania. ACM Press.
157. Boutilier C. and Brafman R. I. (2001) Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14, p. 105–136.
158. Boutilier C., Dearden R., and Goldszmidt M. (2000) Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121, p. 49–107.
159. Boutilier C., Reiter R., and Price B. (2001) Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, p. 467–472, Seattle. Morgan Kaufmann.
160. Boutilier C., Reiter R., Soutchanski M., and Thrun S. (2000) Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, p. 355–362, Austin, Texas. AAAI Press.

161. Box G. E. P. (1957) Evolutionary operation: A method of increasing industrial productivity. *Applied Statistics*, 6, p. 81–101.
162. Boyan J. A. (2002) Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2–3), p. 233–246.
163. Boyan J. A. and Moore A. W. (1998) Learning evaluation functions for global optimization and Boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin. AAAI Press.
164. Boyen X., Friedman N., and Koller D. (1999) Discovering the hidden structure of complex dynamic systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, Stockholm. Morgan Kaufmann.
165. Boyer R. S. and Moore J. S. (1979) *A Computational Logic*. Academic Press, New York.
166. Boyer R. S. and Moore J. S. (1984) Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly*, 91(3), p. 181–189.
167. Brachman R. J. (1979) On the epistemological status of semantic networks. In Findler N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, p. 3–50. Academic Press, New York.
168. Brachman R. J., Fikes R. E., and Levesque H. J. (1983) Krypton: A functional approach to knowledge representation. *Computer*, 16(10), p. 67–73.
169. Brachman R. J. and Levesque H. J. (Eds.) (1985) *Readings in Knowledge Representation*. Morgan Kaufmann, San Mateo, California.
170. Bradtko S. J. and Barto A. G. (1996) Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22, p. 33–57.
171. Brafman R. I. and Tennenholtz M. (2000) A near optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence*, 121, p. 31–47.
172. Braitenberg V. (1984) *Vehicles: Experiments in Synthetic Psychology*. MIT Press.
173. Bransford J. and Johnson M. (1973) Consideration of some problems in comprehension. In Chase W. G. (Ed.), *Visual Information Processing*. Academic Press, New York.
174. Bratko I. (1986) *Prolog Programming for Artificial Intelligence* (1st edition). Addison-Wesley, Reading, Massachusetts.
175. Bratko I. (2001) *Prolog Programming for Artificial Intelligence* (Third edition). Addison-Wesley, Reading, Massachusetts.
176. Bratman M. E. (1987) *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts.
177. Bratman M. E. (1992) Planning and the stability of intention. *Minds and Machines*, 2(1), p. 1–16.
178. Breese J. S. and Heckerman D. (1996) Decision-theoretic troubleshooting: A framework for repair and experiment. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, p. 124–132, Portland, Oregon. Morgan Kaufmann.
179. Breiman L. (1996) Bagging predictors. *Machine Learning*, 26(2), p. 123–140.
180. Breiman L., Friedman J., Olshen R. A., and Stone P. J. (1984) *Classification and Regression Trees*. Wadsworth International Group, Belmont, California.
181. Brelaz D. (1979) New methods to color the vertices of a graph. *Communications of the Association for Computing Machinery*, 22(4), p. 251–256.
182. Brent R. P. (1973) *Algorithms for minimization without derivatives*. Prentice-Hall, Upper Saddle River, New Jersey.
183. Bresnan J. (1982) *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.
184. Brewka G., Dix J., and Konolige K. (1997) *Nonmonotonic Reasoning: An Overview*. CSLI Publications, Stanford, California.

185. Bridle J. S. (1990) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Soulié, F. and Héault J. (Eds.), *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, Berlin.
186. Briggs R. (1985) Knowledge representation in Sanskrit and artificial intelligence. *AI Magazine*, 6(1), p. 32–39.
187. Brin S. and Page L. (1998) The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh World Wide Web Conference*, Brisbane, Australia.
188. Broadbent D. E. (1958) *Perception and communication*. Pergamon, Oxford, UK.
189. Brooks R. A. (1986) A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2, p. 14–23.
190. Brooks R. A. (1989) Engineering approach to building complete, intelligent beings. *Proceedings of the SPIE — the International Society for Optical Engineering*, 1002, p. 618–625.
191. Brooks R. A. (1990) Elephants don't play chess. *Autonomous Robots*, 6, p. 3–15.
192. Brooks R. A. (1991) Intelligence without representation. *Artificial Intelligence*, 47(1–3), p. 139–159.
193. Brooks R. A. and Lozano-Perez T. (1985) A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man and Cybernetics*, 15(2), p. 224–233.
194. Brown M., Grundy W., Lin D., Cristianini N., Sugnet C., Furey T., Ares M., and Haussler D. (2000) Knowledge-based analysis of microarray gene expression data using support vector machines. In *Proceedings of the national Academy of Sciences*, Vol. 97, p. 262–267.
195. Brown P. F., Cocke J., Della Pietra S. A., Della Pietra V. J., Jelinek F., Mercer R. L., and Roossin P. (1988) A statistical approach to language translation. In *Proceedings of the 12th International Conference on Computational Linguistics*, p. 71–76, Budapest. John von Neumann Society for Computing Sciences.
196. Brown P. F., Della Pietra S. A., Della Pietra V. J., and Mercer R. L. (1993) The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2), p. 263–311.
197. Brownston L., Farrell R., Kant E., and Martin N. (1985) *Programming expert systems in OPS5: An introduction to rule-based programming*. Addison-Wesley, Reading, Massachusetts.
198. Brudno A. L. (1963) Bounds and valuations for shortening the scanning of variations. *Problems of Cybernetics*, 10, p. 225–241.
199. Bruner J. S., Goodnow J. J., and Austin G. A. (1957) *A Study of Thinking*. Wiley, New York.
200. Bryant R. E. (1992) Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3), p. 293–318.
201. Bryson A. E. and Ho Y.-C. (1969) *Applied Optimal Control*. Blaisdell, New York.
202. Buchanan B. G. and Mitchell T. M. (1978) Model-directed learning of production rules. In Waterman D. A. and Hayes-Roth F. (Eds.), *Pattern-Directed Inference Systems*, p. 297–312. Academic Press, New York.
203. Buchanan B. G., Mitchell T. M., Smith R. G., and Johnson C. R. (1978) Models of learning systems. In *Encyclopedia of Computer Science and Technology*, Vol. 11. Dekker, New York.
204. Buchanan B. G. and Shortliffe E. H. (Eds.) (1984) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts.
205. Buchanan B. G., Sutherland G. L., and Feigenbaum E. A. (1969) Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. In Meltzer B., Michie D., and Swann M. (Eds.), *Machine Intelligence 4*, p. 209–254. Edinburgh University Press, Edinburgh, Scotland.
206. Bundy A. (1999) A survey of automated deduction. In Wooldridge M. J. and Veloso M. (Eds.), *Artificial intelligence today: Recent trends and developments*, p. 153–174. Springer-Verlag, Berlin.

207. Bunt H. C. (1985) The formal representation of (quasi-) continuous concepts. In Hobbs J. R. and Moore R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 2, p. 37–70. Ablex, Norwood, New Jersey.
208. Burgard W., Cremers A. B., Fox D., Hähnel D., Lakemeyer G., Schulz D., Steiner W., and Thrun S. (1999) Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1–2), p. 3–55.
209. Buro M. (2002) Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1–2), p. 85–99.
210. Burstall R. M. (1974) Program proving as hand simulation with a little induction. In *Information Processing '74*, p. 308–312. Elsevier/North-Holland, Amsterdam, London, New York.
211. Burstall R. M. and Darlington J. (1977) A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1), p. 44–67.
212. Burstein J., Leacock C., and Swartz R. (2001) Automated evaluation of essays and short answers. In *Fifth International Computer Assisted Assessment (CAA) Conference*, Loughborough U.K. Loughborough University.
213. Bylander T. (1992) Complexity results for serial decomposability. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, p. 729–734, San Jose. AAAI Press.
214. Bylander T. (1994) The computational complexity of propositional strips planning. *Artificial Intelligence*, 69, p. 165–204.
215. Calvanese D., Lenzerini M., and Nardi D. (1999) Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, 11, p. 199–240.
216. Campbell M. S., Hoane A. J., and Hsu F.-H. (2002) Deep Blue. *Artificial Intelligence*, 134(1–2), p. 57–83.
217. Canny J. and Reif J. (1987) New lower bound techniques for robot motion planning problems. *IEEE Symposium on Foundations of Computer Science*, p. 39–48.
218. Canny J. (1986) A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8, p. 679–698.
219. Canny J. (1988) *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Massachusetts.
220. Carbonell J. G. (1983) Derivational analogy and its role in problem solving. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, p. 64–69, Washington, DC. Morgan Kaufmann.
221. Carbonell J. G., Knoblock C. A., and Minton S. (1989) PRODIGY: An integrated architecture for planning and learning. Technical report CMU-CS-p. 89–189, Computer Science Department, Carnegie-Mellon University, Pittsburgh.
222. Carbonell J. R. and Collins A. M. (1973) Natural semantics in artificial intelligence. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, p. 344–351, Stanford, California. IJCAII.
223. Carnap R. (1928) *Der logische Aufbau der Welt*. Weltkreis-verlag, Berlin-Schlachtensee. Перевод на английский: Carnap R. (1967) *The Logical Structure of the World & Pseudoproblems in Philosophy*. University of California Press, Berkeley.
224. Carnap R. (1948) On the application of inductive logic. *Philosophy and Phenomenological Research*, 8, p. 133–148.
225. Carnap R. (1950) *Logical Foundations of Probability*. University of Chicago Press, Chicago.
226. Carrasco R. C., Oncina J., and Calera J. (1998) *Stochastic Inference of Regular Tree Languages*, Vol. 1433 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.
227. Cassandra A. R., Kaelbling L. P., and Littman M. L. (1994) Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1023–1028, Seattle. AAAI Press.
228. Ceri S., Gottlob G., and Tanca L. (1990) *Logic programming and databases*. Springer-Verlag, Berlin.

229. Chakrabarti P. P., Ghose S., Acharya A., and de Sarkar S. C. (1989) Heuristic search in restricted memory. *Artificial Intelligence*, 41(2), p. 197–222.
230. Chan W. P., Prete F., and Dickinson M. H. (1998) Visual input to the efferent control system of a fly's "gyroscope". *Science*, 289, p. 289–292.
231. Chandra A. K. and Harel D. (1980) Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2), p. 156–178.
232. Chandra A. K. and Merlin P. M. (1977) Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, p. 77–90, New York. ACM Press.
233. Chang C.-L. and Lee R. C.-T. (1973) *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.
234. Chapman D. (1987) Planning for conjunctive goals. *Artificial Intelligence*, 32(3), p. 333–377.
235. Charniak E. (1993) *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.
236. Charniak E. (1996) Tree-bank grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, p. 1031–1036, Portland, Oregon. AAAI Press.
237. Charniak E. (1997) Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, p. 598–603, Providence, Rhode Island. AAAI Press.
238. Charniak E. and Goldman R. (1992) A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1), p. 53–79.
239. Charniak E. and McDermott D. (1985) *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts.
240. Charniak E., Riesbeck C., McDermott D., and Meehan J. (1987) *Artificial Intelligence Programming* (2nd edition). Lawrence Erlbaum Associates, Potomac, Maryland.
241. Chatfield C. (1989) *The Analysis of Time Series: An Introduction* (4th edition). Chapman and Hall, London.
242. Cheeseman P. (1985) In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, p. 1002–1009, Los Angeles. Morgan Kaufmann.
243. Cheeseman P. (1988) An inquiry into computer understanding. *Computational Intelligence*, 4(1), p. 58–66.
244. Cheeseman P., Kanefsky B., and Taylor W. (1991) Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, p. 331–337, Sydney. Morgan Kaufmann.
245. Cheeseman P., Self M., Kelly J., and Stutz J. (1988) Bayesian classification. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Vol. 2, p. 607–611, St. Paul, Minnesota. Morgan Kaufmann.
246. Cheeseman P. and Stutz J. (1996) Bayesian classification (AutoClass): Theory and results. In Fayyad U., Piatetsky-Shapiro G., Smyth P., and Uthurusamy R. (Eds.), *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, Menlo Park, California.
247. Cheng J. and Druzdzel M. J. (2000) AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13, p. 155–188.
248. Cheng J., Greiner R., Kelly J., Bell D. A., and Liu W. (2002) Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137, p. 43–90.
249. Chierchia G. and McConnell-Ginet S. (1990) *Meaning and Grammar*. MIT Press, Cambridge, Massachusetts.
250. Chomsky N. (1956) Three models for the description of language. *IRE Transactions on Information Theory*, 2(3), p. 113–124.

251. Chomsky N. (1957) *Syntactic Structures*. Mouton, The Hague and Paris.
252. Chomsky N. (1980) Rules and representations. *The Behavioral and Brain Sciences*, 3, p. 1–61.
253. Choset H. (1996) *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. Ph.D. thesis, California Institute of Technology.
254. Chung K. L. (1979) *Elementary Probability Theory with Stochastic Processes* (3rd edition). Springer-Verlag, Berlin.
255. Church A. (1936) A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1, p. 40–41 and 101–102.
256. Church K. and Patil R. (1982) Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8(3–4), p. 139–149.
257. Church K. and Gale W. A. (1991) A comparison of the enhanced Good–Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5, p. 19–54.
258. Churchland P. M. (1979) *Scientific Realism and the Plasticity of Mind*. Cambridge University Press, Cambridge, UK.
259. Churchland P. M. and Churchland P. S. (1982) Functionalism, qualia, and intentionality. In Biro J. I. and Shahar R. W. (Eds.) *Mind, Brain and Function: Essays in the Philosophy of Mind*, p. 121–145. University of Oklahoma Press, Norman, Oklahoma.
260. Churchland P. S. (1986) *Neurophilosophy: Toward a Unified Science of the Mind–Brain*. MIT Press, Cambridge, Massachusetts.
261. Cimatti A., Roveri M., and Traverso P. (1998) Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 875–881, Madison, Wisconsin. AAAI Press.
262. Clark K. L. (1978) Negation as failure. In Gallaire H. and Minker J. (Eds.), *Logic and Data Bases*, p. 293–322. Plenum, New York.
263. Clark P. and Niblett T. (1989) The CN2 induction algorithm. *Machine Learning*, 3, p. 261–283.
264. Clarke A. C. (1968a) *2001: A Space Odyssey*. Signet.
265. Clarke A. C. (1968b) *The world of 2001*. Vogue.
266. Clarke E. and Grumberg O. (1987) Research on automatic verification of finite-state concurrent systems. *Annual Review of Computer Science*, 2, p. 269–290.
267. Clarke E., Grumberg O., and Peled D. (1999) *Model Checking*. MIT Press, Cambridge, Massachusetts.
268. Clarke M. R. B. (Ed.) (1977) *Advances in Computer Chess 1*. Edinburgh University Press, Edinburgh, Scotland.
269. Clearwater S. H. (Ed.) (1996) *Market-Based Control*. World Scientific, Singapore and Teaneck, New Jersey.
270. Clocksin W. F. and Mellish C. S. (1994) *Programming in Prolog* (4th edition). Springer-Verlag, Berlin.
271. Clowes M. B. (1971) On seeing things. *Artificial Intelligence*, 2(1), p. 79–116.
272. Cobham A. (1964) The intrinsic computational difficulty of functions. In Bar-Hillel Y. (Ed.), *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, p. 24–30, Jerusalem. Elsevier/North-Holland.
273. Cobley P. (1997) *Introducing Semiotics*. Totem Books, New York.
274. Cohen J. (1988) A view of the origins and development of PROLOG. *Communications of the Association for Computing Machinery*, 31, p. 26–36.
275. Cohen P. R. (1995) *Empirical methods for artificial intelligence*. MIT Press, Cambridge, Massachusetts.
276. Cohen P. R. and Levesque H. J. (1990) Intention is choice with commitment. *Artificial Intelligence*, 42(2–3), p. 213–261.

277. Cohen P. R., Morgan J., and Pollack M. E. (1990) *Intentions in Communication*. MIT Press, Cambridge, Massachusetts.
278. Cohen P. R. and Perrault C. R. (1979) Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3), p. 177–212.
279. Cohen W. W. and Page C. D. (1995) Learnability in inductive logic programming: Methods and results. *New Generation Computing*, 13(3–4), p. 369–409.
280. Cohn A. G., Bennett B., Goody J. M., and Gotts N. (1997) RCC: A calculus for region based qualitative spatial reasoning. *GeoInformatica*, 1, p. 275–316.
281. Collins M. J. (1996) A new statistical parser based on bigram lexical dependencies. In Joshi A. K. and Palmer M. (Eds.), *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, p. 184–191, San Francisco. Morgan Kaufmann Publishers.
282. Collins M. J. (1999) *Head-driven Statistical Models for Natural Language Processing*. Ph.D. thesis, University of Pennsylvania.
283. Collins M. and Duffy K. (2002) New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the ACL*.
284. Colmérauer A. (1975) Les grammaires de métamorphose. Tech. rep., Groupe d'Intelligence Artificielle, Université de Marseille-Luminy.
285. Colmérauer A., Kanoui H., Pasero R., and Roussel P. (1973) Un système de communication homme-machine en Français. Rapport, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.
286. Condon J. H. and Thompson K. (1982) Belle chess hardware. In Clarke M. R. B. (Ed.), *Advances in Computer Chess 3*, p. 45–54. Pergamon, Oxford, UK.
287. Congdon C. B., Huber M., Kortenkamp D., Bidlack C., Cohen C., Huffman S., Koss F., Raschke U., and Weymouth T. (1992) CARMEL versus Flakey: A comparison of two robots. Tech. rep. Papers from the AAAI Robot Competition, RC-92-01, American Association for Artificial Intelligence, Menlo Park, CA.
288. Connell J. (1989) *A Colony Architecture for an Artificial Creature*. Ph.D. thesis, Artificial Intelligence Laboratory, MIT, Cambridge, MA. Предоставляется также под названием AI Technical Report 1151.
289. Cook S. A. (1971) The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, p. 151–158, New York. ACM Press.
290. Cook S. A. and Mitchell D. (1997) Finding hard instances of the satisfiability problem: A survey. In Du D., Gu J., and Pardalos P. (Eds.), *Satisfiability problems: Theory and applications*. American Mathematical Society, Providence, Rhode Island.
291. Cooper G. (1990) The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42, p. 393–405.
292. Cooper G. and Herskovits E. (1992) A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9, p. 309–347.
293. Copeland J. (1993) *Artificial Intelligence: A Philosophical Introduction*. Blackwell, Oxford, UK.
294. Copernicus (1543) *De Revolutionibus Orbium Coelestium*. Apud Ioh. Petreium, Nuremberg.
295. Cormen T. H., Leiserson C. E., and Rivest R. (1990) *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts.
296. Cortes C. and Vapnik V. N. (1995) Support vector networks. *Machine Learning*, 20, p. 273–297.
297. Cournot A. (Ed.) (1838) *Recherches sur les principes mathématiques de la théorie des richesses*. L. Hachette, Paris.
298. Covington M. A. (1994) *Natural Language Processing for Prolog Programmers*. Prentice-Hall, Upper Saddle River, New Jersey.
299. Cowan J. D. and Sharp D. H. (1988a) Neural nets. *Quarterly Reviews of Biophysics*, 21, p. 365–427.

300. Cowan J. D. and Sharp D. H. (1988b) Neural nets and artificial intelligence. *Daedalus*, 117, p. 85–121.
301. Cox I. (1993) A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, 10, p. 53–66.
302. Cox I. and Hingorani S. L. (1994) An efficient implementation and evaluation of Reid's multiple hypothesis tracking algorithm for visual tracking. In *Proceedings of the 12th International Conference on Pattern Recognition*, Vol. 1, p. 437–442, Jerusalem, Israel. International Association for Pattern Recognition (IAPR).
303. Cox I. and Wilfong G. T. (Eds.) (1990) *Autonomous Robot Vehicles*. Springer Verlag, Berlin.
304. Cox R. T. (1946) Probability, frequency, and reasonable expectation. *American Journal of Physics*, 14(1), p. 1–13.
305. Craig J. (1989) *Introduction to Robotics: Mechanics and Control (2nd Edition)*. Addison-Wesley Publishing, Inc., Reading, MA.
306. Craik K. J. (1943) *The Nature of Explanation*. Cambridge University Press, Cambridge, UK.
307. Crawford J. M. and Auton L. D. (1993) Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, p. 21–27, Washington, DC. AAAI Press.
308. Cristianini N. and Schölkopf B. (2002) Support vector machines and kernel methods: The new generation of learning machines. *AI Magazine*, 23(3), p. 31–41.
309. Cristianini N. and Shawe-Taylor J. (2000) *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, UK.
310. Crockett L. (1994) *The Turing Test and the Frame Problem: AI's Mistaken Understanding of Intelligence*. Ablex, Norwood, New Jersey.
311. Cross S. E. and Walker E. (1994) Dart: Applying knowledge based planning and scheduling to crisis action planning. In Zweben M. and Fox M. S. (Eds.), *Intelligent Scheduling*, p. 711–729. Morgan Kaufmann, San Mateo, California.
312. Cruse D. A. (1986) *Lexical Semantics*. Cambridge University Press.
313. Culberson J. and Schaeffer J. (1998) Pattern databases. *Computational Intelligence*, 14(4), p. 318–334.
314. Cullingford R. E. (1981) Integrating knowledge sources for computer “understanding” tasks. *IEEE Transactions on Systems, Man and Cybernetics (SMC)*, 11.
315. Cussens J. and Dzeroski S. (2000) *Learning Language in Logic*, Vol. 1925 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.
316. Cybenko G. (1988) Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, Massachusetts.
317. Cybenko G. (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems*, 2, p. 303–314.
318. Daganzo C. (1979) *Multinomial probit: The theory and its application to demand forecasting*. Academic Press, New York.
319. Dagum P. and Luby M. (1993) Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1), p. 141–153.
320. Dahl O.-J., Myrhaug B., and Nygaard K. (1970) (Simula 67) common base language. Tech. rep. N. S-22, Norsk Regnesentral (Norwegian Computing Center), Oslo.
321. Dale R., Moisl H., and Somers H. (2000) *Handbook of Natural Language Processing*. Marcel Dekker, New York.
322. Dantzig G. B. (1949) Programming of interdependent activities: II. Mathematical model. *Econometrica*, 17, p. 200–211.
323. Darwiche A. (2001) Recursive conditioning. *Artificial Intelligence*, 126, p. 5–41.

324. Darwiche A. and Ginsberg M. L. (1992) A symbolic generalization of probability theory. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, p. 622–627, San Jose. AAAI Press.
325. Darwin C. (1859) *On The Origin of Species by Means of Natural Selection*. J. Murray, London.
326. Darwin C. (1871) *Descent of Man*. J. Murray.
327. Dasgupta P., Chakrabarti P. P., and DeSarkar S. C. (1994) Agent searching in a tree and the optimality of iterative deepening. *Artificial Intelligence*, 71, p. 195–208.
328. Davidson D. (1980) *Essays on Actions and Events*. Oxford University Press, Oxford, UK.
329. Davies T. R. (1985) Analogy. Informal note IN-CSLI-85-4, Center for the Study of Language and Information (CSLI), Stanford, California.
330. Davies T. R. and Russell S. J. (1987) A logical approach to reasoning by analogy. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Vol. 1, p. 264–270, Milan. Morgan Kaufmann.
331. Davis E. (1986) *Representing and Acquiring Geographic Knowledge*. Pitman and Morgan Kaufmann, London and San Mateo, California.
332. Davis E. (1990) *Representations of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, California.
333. Davis K. H., Biddulph R., and Balashek S. (1952) Automatic recognition of spoken digits. *Journal of the Acoustical Society of America*, 24(6), p. 637–642.
334. Davis M. (1957) A computer program for Presburger's algorithm. In Robinson A. (Ed.), *Proving Theorems (as Done by Man, Logician, or Machine)*, p. 215–233, Cornell University, Ithaca, New York. Communications Research Division, Institute for Defense Analysis. Proceedings of the Summer Institute for Symbolic Logic. Second edition; publication date is 1960.
335. Davis M., Logemann G., and Loveland D. (1962) A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5, p. 394–397.
336. Davis M. and Putnam H. (1960) A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3), p. 201–215.
337. Davis R. and Lenat D. B. (1982) *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, New York.
338. Dayan P. (1992) The convergence of TD( $\lambda$ ) for general  $\lambda$ . *Machine Learning*, 8(3–4), p. 341–362.
339. Dayan P. and Abbott L. F. (2001) *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Cambridge, Massachusetts.
340. de Dombal F. T., Leaper D. J., Horrocks J. C., and Staniland J. R. (1974) Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal*, 1, p. 376–380.
341. de Dombal F. T., Staniland J. R., and Clamp S. E. (1981) Geographical variation in disease presentation. *Medical Decision Making*, 1, p. 59–69.
342. de Finetti B. (1937) Le prévision: ses lois logiques, ses sources subjectives. *Ann. Inst. Poincaré*, 7, p. 1–68.
343. de Freitas J. F. G., Niranjan M., and Gee A. H. (2000) Sequential Monte Carlo methods to train neural network models. *Neural Computation*, 12(4), p. 933–953.
344. de Kleer J. (1975) Qualitative and quantitative knowledge in classical mechanics. Tech. rep. AI-TR-352, MIT Artificial Intelligence Laboratory.
345. de Kleer J. (1986a) An assumption-based TMS. *Artificial Intelligence*, 28(2), p. 127–162.
346. de Kleer J. (1986b) Extending the ATMS. *Artificial Intelligence*, 28(2), p. 163–196.
347. de Kleer J. (1986c) Problem solving with the ATMS. *Artificial Intelligence*, 28(2), p. 197–224.

348. de Kleer J. (1989) A comparison of ATMS and CSP techniques. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 1, p. 290–296, Detroit. Morgan Kaufmann.
349. de Kleer J. and Brown J. S. (1985) A qualitative physics based on confluentes. In Hobbs J. R. and Moore R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 4, p. 109–183. Ablex, Norwood, New Jersey.
350. de Marcken C. (1996) *Unsupervised Language Acquisition*. Ph.D. thesis, MIT.
351. De Morgan A. (1864) On the syllogism IV and on the logic of relations. *Cambridge Philosophical Transactions*, x, p. 331–358.
352. De Raedt L. (1992) *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press, New York.
353. de Saussure F. (1910; повторно опубликовано в 1993) *Lectures on General Linguistics*. Pergamon Press, Oxford, UK.
354. Deacon T. W. (1997) *The symbolic species: The co-evolution of language and the brain*. W. W. Norton, New York.
355. Deale M., Yvanovich M., Schnitzius D., Kautz D., Carpenter M., Zweben M., Davis G., and Daun B. (1994) The space shuttle ground processing scheduling system. In Zweben M. and Fox M. (Eds.), *Intelligent Scheduling*, p. 423–449. Morgan Kaufmann, San Mateo, California.
356. Dean T., Basye K., Chekaluk R., and Hyun S. (1990) Coping with uncertainty in a control system for navigation and exploration. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, p. 1010–1015, Boston. MIT Press.
357. Dean T. and Boddy M. (1988) An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, p. 49–54, St. Paul, Minnesota. Morgan Kaufmann.
358. Dean T., Firby R. J., and Miller D. (1990) Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 6(1), 381–398.
359. Dean T., Kaelbling L. P., Kirman J., and Nicholson A. (1993) Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, p. 574–579, Washington, DC. AAAI Press.
360. Dean T. and Kanazawa K. (1989a) A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, p. 985–990, Detroit. Morgan Kaufmann.
361. Dean T. and Kanazawa K. (1989b) A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3), p. 142–150.
362. Dean T., Kanazawa K., and Shewchuk J. (1990) Prediction, observation and estimation in planning and control. In *5th IEEE International Symposium on Intelligent Control*, Vol. 2, p. 645–650, Los Alamitos, CA. IEEE Computer Society Press.
363. Dean T. and Wellman M. P. (1991) *Planning and Control*. Morgan Kaufmann, San Mateo, California.
364. Debevec P., Taylor C., and Malik J. (1996) Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics (SIGGRAPH)*, p. 11–20.
365. Debreu G. (1960) Topological methods in cardinal utility theory. In Arrow K. J., Karlin S., and Suppes P. (Eds.), *Mathematical Methods in the Social Sciences, 1959*. Stanford University Press, Stanford, California.
366. Dechter R. (1990a) Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41, p. 273–312.
367. Dechter R. (1990b) On the expressiveness of networks with hidden variables. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, p. 379–385, Boston. MIT Press.

368. Dechter R. (1992) Constraint networks. In Shapiro S. (Ed.), *Encyclopedia of Artificial Intelligence* (2nd edition), p. 276–285. Wiley and Sons, New York.
369. Dechter R. (1999) Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113, p. 41–85.
370. Dechter R. and Frost D. (1999) Backtracking algorithms for constraint satisfaction problems. Tech. rep., Department of Information and Computer Science, University of California, Irvine.
371. Dechter R. and Pearl J. (1985) Generalized best-first search strategies and the optimality of A\*. *Journal of the Association for Computing Machinery*, 32(3), p. 505–536.
372. Dechter R. and Pearl J. (1987) Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1), p. 1–38.
373. Dechter R. and Pearl J. (1989) Tree clustering for constraint networks. *Artificial Intelligence*, 38(3), p. 353–366.
374. DeCoste D. and Scholkopf B. (2002) Training invariant support vector machines. *Machine Learning*, 46(1), p. 161–190.
375. Dedekind R. (1888) *Was sind und was sollen die Zahlen*. Braunschweig, Germany.
376. Deerwester S. C., Dumais S. T., Landauer T. K., Furnas G. W., and Harshman R. A. (1990) Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6), p. 391–407.
377. DeGroot M. H. (1970) *Optimal Statistical Decisions*. McGraw-Hill, New York.
378. DeGroot M. H. (1989) *Probability and Statistics* (2nd edition). Addison-Wesley, Reading, Massachusetts.
379. DeJong G. (1981) Generalizations based on explanations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, p. 67–69, Vancouver, British Columbia. Morgan Kaufmann.
380. DeJong G. (1982) An overview of the FRUMP system. In Lehnert W. and Ringle M. (Eds.), *Strategies for Natural Language Processing*, p. 149–176. Lawrence Erlbaum, Potomac, Maryland.
381. DeJong G. and Mooney R. (1986) Explanation-based learning: An alternative view. *Machine Learning*, 1, p. 145–176.
382. Dempster A. P. (1968) A generalization of Bayesian inference. *Journal of the Royal Statistical Society, Series B*, 30(Series B), p. 205–247.
383. Dempster A. P., Laird N., and Rubin D. (1977) Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(Series B), p. 1–38.
384. Denes P. (1959) The design and operation of the mechanical speech recognizer at University College London. *Journal of the British Institution of Radio Engineers*, 19(4), p. 219–234.
385. Deng X. and Papadimitriou C. H. (1990) Exploring an unknown graph. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, p. 355–361, St. Louis. IEEE Computer Society Press.
386. Denis F. (2001) Learning regular languages from simple positive examples. *Machine Learning*, 44(1/2), p. 37–66.
387. Dennett D. C. (1971) Intentional systems. *The Journal of Philosophy*, 68(4), p. 87–106.
388. Dennett D. C. (1978) Why you can't make a computer that feels pain. *Synthese*, 38(3).
389. Dennett D. C. (1984) Cognitive wheels: the frame problem of AI. In Hookway C. (Ed.), *Minds, Machines, and Evolution: Philosophical Studies*, p. 129–151. Cambridge University Press, Cambridge, UK.
390. Deo N. and Pang C.-Y. (1984) Shortest path algorithms: Taxonomy and annotation. *Networks*, 14(2), p. 275–323.
391. Descartes R. (1637) Discourse on method. In Cottingham J., Stoothoff R., and Murdoch D. (Eds.), *The Philosophical Writings of Descartes*, Vol. I. Cambridge University Press, Cambridge, UK.

392. Descotte Y. and Latombe J.-C. (1985) Making compromises among antagonist constraints in a planner. *Artificial Intelligence*, 27, p. 183–217.
393. Devroye L. (1987) *A course in density estimation*. Birkhauser, Boston.
394. Devroye L., Gyorfi L., and Lugosi G. (1996) *A probabilistic theory of pattern recognition*. Springer-Verlag, Berlin.
395. Dickmanns E. D. and Zapp A. (1987) Autonomous high speed road vehicle guidance by computer vision. In Isermann R. (Ed.), *Automatic Control — World Congress, 1987: Selected Papers from the 10th Triennial World Congress of the International Federation of Automatic Control*, p. 221–226, Munich. Pergamon.
396. Dietterich T. (1990) Machine learning. *Annual Review of Computer Science*, 4, p. 255–306.
397. Dietterich T. (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13, p. 227–303.
398. DiGioia A. M., Kanade T., and Wells P. (1996) Final report of the second international workshop on robotics and computer assisted medical interventions. *Computer Aided Surgery*, 2, p. 69–101.
399. Dijkstra E. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, p. 269–271.
400. Dissanayake G., Newman P., Clark S., Durrant-Whyte H., and Csorba M. (2001) A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions of Robotics and Automation*, 17(3), p. 229–241.
401. Do M. B. and Kambhampati S. (2001) Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of the European Conference on Planning*, Toledo, Spain. Springer-Verlag.
402. Domingos P. and Pazzani M. (1997) On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29, p. 103–30.
403. Doran C., Egedi D., Hockey B. A., Srinivas B., and Zaidel M. (1994) XTAG system — a wide coverage grammar of English. In Nagao M. (Ed.), *Proceedings of the 15th COLING*, Kyoto, Japan.
404. Doran J. and Michie D. (1966) Experiments with the graph traverser program. *Proceedings of the Royal Society of London*, 294, Series A, p. 235–259.
405. Dorf R. C. and Bishop R. H. (1999) *Modern Control Systems*. Addison-Wesley, Reading, Massachusetts.
406. Doucet A. (1997) *Monte Carlo methods for Bayesian estimation of hidden Markov models: Application to radiation signals*. Ph.D. thesis, Université de Paris-Sud, Orsay, France.
407. Dowling W. F. and Gallier J. H. (1984) Linear-time algorithms for testing the satisfiability of propositional Horn formulas. *Journal of Logic Programming*, 1, p. 267–284.
408. Dowty D., Wall R., and Peters S. (1991) *Introduction to Montague Semantics*. D. Reidel, Dordrecht, Netherlands.
409. Doyle J. (1979) A truth maintenance system. *Artificial Intelligence*, 12(3), p. 231–272.
410. Doyle J. (1983) What is rational psychology? Toward a modern mental philosophy. *AI Magazine*, 4(3), p. 50–53.
411. Doyle J. and Patil R. (1991) Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3), p. 261–297.
412. Drabble B. (1990) Mission scheduling for spacecraft: Diaries of T-SCHED. In *Expert Planning Systems*, p. 76–81, Brighton, UK. Institute of Electrical Engineers.
413. Draper D., Hanks S., and Weld D. S. (1994) Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on AI Planning Systems*, p. 31–36, Chicago. Morgan Kaufmann.
414. Dreyfus H. L. (1972) *What Computers Can't Do: A Critique of Artificial Reason*. Harper and Row, New York.

415. Dreyfus H. L. (1992) *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press, Cambridge, Massachusetts.
416. Dreyfus H. L. and Dreyfus S. E. (1986) *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. Blackwell, Oxford, UK.
417. Dreyfus S. E. (1969) An appraisal of some shortest-paths algorithms. *Operations Research*, 17, p. 395–412.
418. Du D., Gu J., and Pardalos P. M. (Eds.) (1999) *Optimization methods for logical inference*. American Mathematical Society, Providence, Rhode Island.
419. Dubois D. and Prade H. (1994) A survey of belief revision and updating rules in various uncertainty models. *International Journal of Intelligent Systems*, 9(1), p. 61–100.
420. Duda R. O., Gaschnig J., and Hart P. E. (1979) Model design in the Prospector consultant system for mineral exploration. In Michie D. (Ed.), *Expert Systems in the Microelectronic Age*, p. 153–167. Edinburgh University Press, Edinburgh, Scotland.
421. Duda R. O. and Hart P. E. (1973) *Pattern classification and scene analysis*. Wiley, New York.
422. Duda R. O., Hart P. E., and Stork D. G. (2001) *Pattern Classification*. Wiley, New York.
423. Dudek G. and Jenkin M. (2000) *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge CB2 2RU, UK.
424. Durfee E. H. and Lesser V. R. (1989) Negotiating task decomposition and allocation using partial global planning. In Huhns M. and Gasser L. (Eds.), *Distributed AI*, Vol. 2. Morgan Kaufmann, San Mateo, California.
425. Dyer M. (1983) *In-Depth Understanding*. MIT Press, Cambridge, Massachusetts.
426. Dzeroski S., Muggleton S. H., and Russell S. J. (1992) PAC-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*, p. 128–135, Pittsburgh, Pennsylvania. ACM Press.
427. Earley J. (1970) An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2), p. 94–102.
428. Ebeling C. (1987) *All the Right Moves*. MIT Press, Cambridge, Massachusetts.
429. Eco U. (1979) *Theory of Semiotics*. Indiana University Press, Bloomington, Indiana.
430. Edmonds J. (1965) Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17, p. 449–467.
431. Edwards P. (Ed.) (1967) *The Encyclopedia of Philosophy*. Macmillan, London.
432. Eiter T., Leone N., Mateis C., Pfeifer G., and Scarcello F. (1998) The KR system dlv: Progress report, comparisons and benchmarks. In Cohn A., Schubert L., and Shapiro S. (Eds.), *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, p. 406–417, Trento, Italy.
433. Elhadad M. (1993) FUF: The universal unifier — user manual. Technical report, Ben Gurion University of the Negev, Be'er Sheva, Israel.
434. Elkan C. (1993) The paradoxical success of fuzzy logic. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, p. 698–703, Washington, DC. AAAI Press.
435. Elkan C. (1997) Boosting and naive Bayesian learning. Tech. rep., Department of Computer Science and Engineering, University of California, San Diego.
436. Elman J., Bates E., Johnson M., Karmiloff-Smith A., Parisi D., and Plunkett K. (1997) *Rethinking Innateness*. MIT Press, Cambridge, Massachusetts.
437. Empson W. (1953) *Seven Types of Ambiguity*. New Directions, New York.
438. Enderton H. B. (1972) *A Mathematical Introduction to Logic*. Academic Press, New York.
439. Erdmann M. A. and Mason M. (1988) An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4), p. 369–379.
440. Erman L. D., Hayes-Roth F., Lesser V. R., and Reddy R. (1980) The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2), p. 213–253.

441. Ernst H. A. (1961) *MH-1, a Computer-Operated Mechanical Hand*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
442. Ernst M., Millstein T., and Weld D. S. (1997) Automatic SAT-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, p. 1169–1176, Nagoya, Japan. Morgan Kaufmann.
443. Erol K., Hendler J., and Nau D. S. (1994) HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1123–1128, Seattle. AAAI Press.
444. Erol K., Hendler J., and Nau D. S. (1996) Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1), p. 69–93.
445. Etzioni O. (1989) Tractable decision-analytic control. In *Proc. of 1st International Conference on Knowledge Representation and Reasoning*, p. 114–125, Toronto.
446. Etzioni O., Hanks S., Weld D. S., Draper D., Lesh N., and Williamson M. (1992) An approach to planning with incomplete information. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts.
447. Etzioni O. and Weld D. S. (1994) A softbot-based interface to the Internet. *Communications of the Association for Computing Machinery*, 37(7), p. 72–76.
448. Evans T. G. (1968) A program for the solution of a class of geometric-analogy intelligence-test questions. In Minsky M. L. (Ed.), *Semantic Information Processing*, p. 271–353. MIT Press, Cambridge, Massachusetts.
449. Fagin R., Halpern J. Y., Moses Y., and Vardi M. Y. (1995) *Reasoning about Knowledge*. MIT Press, Cambridge, Massachusetts.
450. Fahlman S. E. (1974) A planning system for robot construction tasks. *Artificial Intelligence*, 5(1), p. 1–49.
451. Fahlman S. E. (1979) *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, Massachusetts.
452. Faugeras O. (1992) What can be seen in three dimensions with an uncalibrated stereo rig? In Sandini G. (Ed.), *Proceedings of the European Conference on Computer Vision*, Vol. 588 of *Lecture Notes in Computer Science*, p. 563–578. Springer-Verlag.
453. Faugeras O. (1993) *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts.
454. Faugeras O., Luong Q.-T., and Papadopoulo T. (2001) *The Geometry of Multiple Images*. MIT Press, Cambridge, Massachusetts.
455. Fearing R. S. and Hollerbach J. M. (1985) Basic solid mechanics for tactile sensing. *International Journal of Robotics Research*, 4(3), p. 40–54.
456. Featherstone R. (1987) *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Boston, MA.
457. Feigenbaum E. A. (1961) The simulation of verbal learning behavior. *Proceedings of the Western Joint Computer Conference*, 19, p. 121–131.
458. Feigenbaum E. A., Buchanan B. G., and Lederberg J. (1971) On generality and problem solving: A case study using the DENDRAL program. In Meltzer B. and Michie D. (Eds.), *Machine Intelligence 6*, p. 165–190. Edinburgh University Press, Edinburgh, Scotland.
459. Feigenbaum E. A. and Feldman J. (Eds.) (1963) *Computers and Thought*. McGraw-Hill, New York.
460. Feldman J. and Sproull R. F. (1977) Decision theory and artificial intelligence II: The hungry monkey. Technical report, Computer Science Department, University of Rochester.
461. Feldman J. and Yakimovsky Y. (1974) Decision theory and artificial intelligence I: Semantics-based region analyzer. *Artificial Intelligence*, 5(4), p. 349–371.
462. Fellbaum C. (2001) *Wordnet: An Electronic Lexical Database*. MIT Press, Cambridge, Massachusetts.
463. Feller W. (1971) *An Introductioon to Probability Theory and its Applications*, Vol. 2. John Wiley.

464. Ferraris P. and Giunchiglia E. (2000) Planning as satisability in nondeterministic domains. In *Proceedings of Seventeenth National Conference on Artificial Intelligence*, p. 748–753. AAAI Press.
465. Fikes R. E., Hart P. E., and Nilsson N. J. (1972) Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), p. 251–288.
466. Fikes R. E. and Nilsson N. J. (1971) STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4), p. 189–208.
467. Fikes R. E. and Nilsson N. J. (1993) STRIPS, a retrospective. *Artificial Intelligence*, 59(1–2), p. 227–232.
468. Findlay J. N. (1941) Time: A treatment of some puzzles. *Australasian Journal of Psychology and Philosophy*, 19(3), p. 216–235.
469. Finney D. J. (1947) *Probit analysis: A statistical treatment of the sigmoid response curve*. Cambridge University Press, Cambridge, UK.
470. Firby J. (1994) Task networks for controlling continuous processes. In Hammond K. (Ed.), *Proceedings of the Second International Conference on AI Planning Systems*, p. 49–54, Menlo Park, CA. AAAI Press.
471. Firby R. J. (1996) Modularity issues in reactive planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, p. 78–85, Edinburgh, Scotland. AAAI Press.
472. Fischer M. J. and Ladner R. E. (1977) Propositional modal logic of programs. In *Proceedings of the 9th ACM Symposium on the Theory of Computing*, p. 286–294, New York. ACM Press.
473. Fisher R. A. (1922) On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, Series A* 222, p. 309–368.
474. Fix E. and Hodges J. L. (1951) Discriminatory analysis — nonparametric discrimination: Consistency properties. Tech. rep. 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas.
475. Fogel D. B. (2000) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, New Jersey.
476. Fogel L. J., Owens A. J., and Walsh M. J. (1966) *Artificial Intelligence through Simulated Evolution*. Wiley, New York.
477. Forbes J. (2002) *Learning Optimal Control for Autonomous Vehicles*. Ph.D. thesis, University of California, Berkeley.
478. Forbus K. D. (1985) The role of qualitative dynamics in naive physics. In Hobbs J. R. and Moore R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 5, p. 185–226. Ablex, Norwood, New Jersey.
479. Forbus K. D. and de Kleer J. (1993) *Building Problem Solvers*. MIT Press, Cambridge, Massachusetts.
480. Ford K. M. and Hayes P. J. (1995) Turing Test considered harmful. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, p. 972–977, Montreal. Morgan Kaufmann.
481. Forestier J.-P. and Varaiya P. (1978) Multilayer control of large Markov chains. *IEEE Transactions on Automatic Control*, 23(2), p. 298–304.
482. Forgy C. (1981) OPS5 user's manual. Technical report CMU-CS-81-135, Computer Science Department, Carnegie-Mellon University, Pittsburgh.
483. Forgy C. (1982) A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1), p. 17–37.
484. Forsyth D. and Zisserman A. (1991) Reflections on shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 13(7), p. 671–679.
485. Fortescue M. D. (1984) *West Greenlandic*. Croom Helm, London.
486. Foster D. W. (1989) *Elegy by W. W.: A Study in Attribution*. Associated University Presses, Cranbury, New Jersey.

487. Fourier J. (1827) Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824; partie mathématique. *Histoire de l'Académie Royale des Sciences de France*, 7, xlvii–lv.
488. Fox D., Burgard W., Dellaert F., and Thrun S. (1999) Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Orlando, FL. AAAI.
489. Fox M. S. (1990) Constraint-guided scheduling: A short history of research at CMU. *Computers in Industry*, 14(1–3), p. 79–88.
490. Fox M. S., Allen B., and Strohm G. (1982) Job shop scheduling: An investigation in constraint-directed reasoning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, p. 155–158, Pittsburgh, Pennsylvania. Morgan Kaufmann.
491. Fox M. S. and Long D. (1998) The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9, p. 367–421.
492. Frakes W. and Baeza-Yates R. (Eds.) (1992) *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Upper Saddle River, New Jersey.
493. Francis S. and Kucera H. (1967) *Computing Analysis of Present-day American English*. Brown University Press, Providence, Rhode Island.
494. Franco J. and Paull M. (1983) Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5, p. 77–87.
495. Frank R. H. and Cook P. J. (1996) *The Winner-Take-All Society*. Penguin, New York.
496. Frege G. (1879) *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, Berlin. Английский перевод опубликован в [1532].
497. Freuder E. C. (1978) Synthesizing constraint expressions. *Communications of the Association for Computing Machinery*, 21(11), p. 958–966.
498. Freuder E. C. (1982) A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29(1), p. 24–32.
499. Freuder E. C. (1985) A sufficient condition for backtrack-bounded search. *Journal of the Association for Computing Machinery*, 32(4), p. 755–761.
500. Freuder E. C. and Mackworth A. K. (Eds.) (1994) *Constraint-based reasoning*. MIT Press, Cambridge, Massachusetts.
501. Freund Y. and Schapire R. E. (1996) Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy. Morgan Kaufmann.
502. Friedberg R. M. (1958) A learning machine: Part I. *IBM Journal*, 2, p. 2–13.
503. Friedberg R. M., Dunham B., and North T. (1959) A learning machine: Part II. *IBM Journal of Research and Development*, 3(3), p. 282–287.
504. Friedman G. J. (1959) Digital simulation of an evolutionary process. *General Systems Yearbook*, 4, p. 171–184.
505. Friedman J., Hastie T., and Tibshirani R. (2000) Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2), p. 337–374.
506. Friedman N. (1998) The Bayesian structural EM algorithm. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, Madison, Wisconsin. Morgan Kaufmann.
507. Friedman N. and Goldszmidt M. (1996) Learning Bayesian networks with local structure. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, p. 252–262, Portland, Oregon. Morgan Kaufmann.
508. Fry D. B. (1959) Theoretical aspects of mechanical speech recognition. *Journal of the British Institution of Radio Engineers*, 19(4), p. 211–218.
509. Fuchs J. J., Gasquet A., Olalainy B., and Currie K. W. (1990) PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*, p. 70–75, Brighton, UK. Institute of Electrical Engineers.

510. Fudenberg D. and Tirole J. (1991) *Game theory*. MIT Press, Cambridge, Massachusetts.
511. Fukunaga A. S., Rabideau G., Chien S., and Yan D. (1997) ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space*, p. 181–187, Tokyo.
512. Fung R. and Chang K. C. (1989) Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, p. 209–220, Windsor, Ontario. Morgan Kaufmann.
513. Gaifman H. (1964) Concerning measures in first order calculi. *Israel Journal of Mathematics*, 2, p. 1–18.
514. Gallaire H. and Minker J. (Eds.) (1978) *Logic and Databases*. Plenum, New York.
515. Gallier J. H. (1986) *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row, New York.
516. Gallo G. and Pallottino S. (1988) Shortest path algorithms. *Annals of Operations Research*, 13, p. 3–79.
517. Gamba A., Gamberini L., Palmieri G., and Sanna R. (1961) Further experiments with PAPA. *Nuovo Cimento Supplemento*, 20(2), p. 221–231.
518. Garding J. (1992) Shape from texture for smooth curved surfaces in perspective projection. *Journal of Mathematical Imaging and Vision*, 2(4), p. 327–350.
519. Gardner M. (1968) *Logic Machines, Diagrams and Boolean Algebra*. Dover, New York.
520. Garey M. R. and Johnson D. S. (1979) *Computers and Intractability*. W. H. Freeman, New York.
521. Gaschnig J. (1977) A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, p. 457, Cambridge, Massachusetts. IJCAII.
522. Gaschnig J. (1979) Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Computer Science Department, Carnegie-Mellon University.
523. Gasser R. (1995) *Efficiently harnessing computational resources for exhaustive search*. Ph.D. thesis, ETH Zürich, Switzerland.
524. Gasser R. (1998) Solving nine men's morris. In Nowakowski R. (Ed.) *Games of No Chance*. Cambridge University Press, Cambridge, UK.
525. Gat E. (1998) Three-layered architectures. In Kortenkamp D., Bonasso R. P., and Murphy R. (Eds.) *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, p. 195–210. MIT Press.
526. Gauss K. F. (1809) *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Sumtibus F. Perthes et I. H. Besser, Hamburg.
527. Gauss K. F. (1829) Beiträge zur theorie der algebraischen gleichungen. Collected in *Werke*, Vol. 3, p. 71–102. K. Gesellschaft Wissenschaft, Göttingen, Germany, 1876.
528. Gawande A. (2002) *Complications: A Surgeon's Notes on an Imperfect Science*. Metropolitan Books, New York.
529. Ge N., Hale J., and Charniak E. (1998) A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, p. 161–171, Montreal. COLING-ACL.
530. Geiger D., Verma T., and Pearl J. (1990) Identifying independence in Bayesian networks. *Networks*, 20(5), p. 507–534.
531. Gelb A. (1974) *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts.
532. Gelernter H. (1959) Realization of a geometry-theorem proving machine. In *Proceedings of an International Conference on Information Processing*, p. 273–282, Paris. UNESCO House.
533. Gelfond M. and Lifschitz V. (1988) Compiling circumscriptive theories into logic programs. In Reinfrank M., de Kleer J., Ginsberg M. L., and Sandewall E. (Eds.), *Non-Monotonic Reasoning: 2nd International Workshop Proceedings*, p. 74–99, Grassau, Germany. Springer-Verlag.
534. Gelman A., Carlin J. B., Stern H. S., and Rubin D. (1995) *Bayesian Data Analysis*. Chapman & Hall, London.

535. Geman S. and Geman D. (1984) Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(6), p. 721–741.
536. Genesereth M. R. (1984) The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1–3), p. 411–436.
537. Genesereth M. R. and Nilsson N. J. (1987) *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.
538. Genesereth M. R. and Nourbakhsh I. (1993) Time-saving tips for problem solving with incomplete information. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, p. 724–730, Washington, DC. AAAI Press.
539. Genesereth M. R. and Smith D. E. (1981) Meta-level architecture. Memo HPP-81-6, Computer Science Department, Stanford University, Stanford, California.
540. Gentner D. (1983) Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7, p. 155–170.
541. Gentzen G. (1934) Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39, p. 176–210; p. 405–431.
542. Georgeff M. P. and Lansky A. L. (1987) Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, p. 677–682, Seattle. Morgan Kaufmann.
543. Gerevini A. and Schubert L. K. (1996) Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, 5, p. 95–137.
544. Gerevini A. and Serina I. (2002) LPG: A planner based on planning graphs with action costs. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, p. 281–290, Menlo Park, California. AAAI Press.
545. Germann U., Jahr M., Knight K., Marcu D., and Yamada K. (2001) Fast decoding and optimal decoding for machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, p. 228–235, Toulouse, France.
546. Gershwin G. (1937) Let's call the whole thing off. Песня.
547. Ghahramani Z. and Jordan M. I. (1997) Factorial hidden Markov models. *Machine Learning*, 29, p. 245–274.
548. Ghallab M., Howe A., Knoblock C. A., and McDermott D. (1998) PDDL — the planning domain definition language. Tech. rep. DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut.
549. Ghallab M. and Laruelle H. (1994) Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, p. 61–67, Chicago. AAAI Press.
550. Giacomo G. D., Lespérance Y., and Levesque H. J. (2000) ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121, p. 109–169.
551. Gibson J. J. (1950) *The Perception of the Visual World*. Houghton Mifflin, Boston.
552. Gibson J. J. (1979) *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston.
553. Gibson J. J., Olum P., and Rosenblatt F. (1955) Parallax and perspective during aircraft landings. *American Journal of Psychology*, 68, p. 372–385.
554. Gilks W. R., Richardson S., and Spiegelhalter D. J. (Eds.) (1996) *Markov chain Monte Carlo in practice*. Chapman and Hall, London.
555. Gilks W. R., Thomas A., and Spiegelhalter D. J. (1994) A language and program for complex Bayesian modelling. *The Statistician*, 43, p. 169–178.
556. Gilmore P. C. (1960) A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, 4, p. 28–35.
557. Ginsberg M. L. (1989) Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4), p. 40–44.

558. Ginsberg M. L. (1993) *Essentials of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.
559. Ginsberg M. L. (1999) GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, p. 584–589, Stockholm. Morgan Kaufmann.
560. Ginsberg M. L., Frank M., Halpin M. P., and Torrance M. C. (1990) Search lessons learned from crossword puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 1, p. 210–215, Boston. MIT Press.
561. Gittins J. C. (1989) *Multi-Armed Bandit Allocation Indices*. Wiley, New York.
562. Glanc A. (1978) On the etymology of the word “robot”. *SIGART Newsletter*, 67, p. 12.
563. Glover F. (1989) Tabu search: 1. *ORSA Journal on Computing*, 1(3), p. 190–206.
564. Glover F. and Laguna M. (Eds.) (1997) *Tabu search*. Kluwer, Dordrecht, Netherlands.
565. Gödel K. (1930) *Über die Vollständigkeit des Logikkalküls*. Ph.D. thesis, University of Vienna.
566. Gödel K. (1931) Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38, p. 173–198.
567. Goebel J., Volk K., Walker H., and Gerbault F. (1989) Automatic classification of spectra from the infrared astronomical satellite (IRAS). *Astronomy and Astrophysics*, 222, p. L5–L8.
568. Gold B. and Morgan N. (2000) *Speech and Audio Signal Processing*. Wiley, New York.
569. Gold E. M. (1967) Language identification in the limit. *Information and Control*, 10, p. 447–474.
570. Golden K. (1998) Leap before you look: Information gathering in the PUCCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, p. 70–77, Pittsburgh, Pennsylvania. AAAI Press.
571. Goldman N. (1975) Conceptual generation. In Schank R. (Ed.), *Conceptual Information Processing*, chap. 6. North-Holland, Amsterdam.
572. Goldman R. and Boddy M. (1996) Expressive planning and explicit knowledge. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, p. 110–117, Edinburgh, Scotland. AAAI Press.
573. Gomes C., Selman B., and Kautz H. (1998) Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 431–437, Madison, Wisconsin. AAAI Press.
574. Good I. J. (1950) Contribution to the discussion of Eliot Slater’s “Statistics for the chess computer and the factor of mobility”. In *Symposium on Information Theory*, p. 199, London. Ministry of Supply.
575. Good I. J. (1961) A causal calculus. *British Journal of the Philosophy of Science*, 11, p. 305–318.
576. Good I. J. (1965) Speculations concerning the first ultraintelligent machine. In Alt F. L. and Rubinoff M. (Eds.), *Advances in Computers*, Vol. 6, p. 31–88. Academic Press, New York.
577. Goodman D. and Keene R. (1997) *Man versus Machine: Kasparov versus Deep Blue*. H3 Publications, Cambridge, Massachusetts.
578. Goodman N. (1954) *Fact, Fiction and Forecast*. University of London Press, London.
579. Goodman N. (1977) *The Structure of Appearance* (3rd edition). D. Reidel, Dordrecht, Netherlands.
580. Gordon M. J., Milner A. J., and Wadsworth C. P. (1979) *Edinburgh LCF*. Springer-Verlag, Berlin.
581. Gordon N. J. (1994) *Bayesian methods for tracking*. Ph.D. thesis, Imperial College, University of London.
582. Gordon N. J., Salmond D. J., and Smith A. F. M. (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140(2), p. 107–113.
583. Gorry G. A. (1968) Strategies for computer-aided diagnosis. *Mathematical Biosciences*, 2(3–4), p. 293–318.
584. Gorry G. A., Kassirer J. P., Essig A., and Schwartz W. B. (1973) Decision analysis as the basis for computer-aided management of acute renal failure. *American Journal of Medicine*, 55, p. 473–484.

585. Gottlob G., Leone N., and Scarcello F. (1999a) A comparison of structural CSP decomposition methods. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, p. 394–399, Stockholm. Morgan Kaufmann.
586. Gottlob G., Leone N., and Scarcello F. (1999b) Hypertree decompositions and tractable queries. In *Proceedings of the 18th ACM International Symposium on Principles of Database Systems*, p. 21–32, Philadelphia. Association for Computing Machinery.
587. Graham S. L., Harrison M. A., and Ruzzo W. L. (1980) An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3), p. 415–462.
588. Grassmann H. (1861) *Lehrbuch der Arithmetik*. Th. Chr. Fr. Enslin, Berlin.
589. Grayson C. J. (1960) Decisions under uncertainty: Drilling decisions by oil and gas operators. Tech. rep., Division of Research, Harvard Business School, Boston.
590. Green B., Wolf A., Chomsky C., and Laugherty K. (1961) BASEBALL: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference*, p. 219–224.
591. Green C. (1969a) Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, p. 219–239, Washington, DC. IJCAII.
592. Green C. (1969b) Theorem-proving by resolution as a basis for question-answering systems. In Meltzer B., Michie D., and Swann M. (Eds.), *Machine Intelligence 4*, p. 183–205. Edinburgh University Press, Edinburgh, Scotland.
593. Green C. and Raphael B. (1968) The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM National Conference*, Washington, DC. ACM Press.
594. Greenblatt R. D., Eastlake D. E., and Crocker S. D. (1967) The Greenblatt chess program. In *Proceedings of the Fall Joint Computer Conference*, p. 801–810. American Federation of Information Processing Societies (AFIPS).
595. Greiner R. (1989) Towards a formal analysis of EBL. In *Proceedings of the Sixth International Machine Learning Workshop*, p. 450–453, Ithaca, NY. Morgan Kaufmann.
596. Grice H. P. (1957) Meaning. *Philosophical Review*, 66, p. 377–388.
597. Grosz B. J., Joshi A. K., and Weinstein S. (1995) Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2), p. 203–225.
598. Grosz B. J. and Sidner C. L. (1986) Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3), p. 175–204.
599. Grosz B. J., Sparck Jones K., and Webber B. L. (Eds.) (1986) *Readings in Natural Language Processing*. Morgan Kaufmann, San Mateo, California.
600. Grove W. and Meehl P. (1996) Comparative efficiency of informal (subjective, impressionistic) and formal (mechanical, algorithmic) prediction procedures: The clinical statistical controversy. *Psychology, Public Policy, and Law*, 2, p. 293–323.
601. Gu J. (1989) *Parallel Algorithms and Architectures for Very Fast AI Search*. Ph.D. thesis, University of Utah.
602. Guard J., Oglesby F., Bennett J., and Settle L. (1969) Semi-automated mathematics. *Journal of the Association for Computing Machinery*, 16, p. 49–62.
603. Guibas L. J., Knuth D. E., and Sharir M. (1992) Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7, 381–413. See also *17th Int. Coll. on Automata, Languages and Programming*, 1990, p. 414–431.
604. Haas A. (1986) A syntactic theory of belief and action. *Artificial Intelligence*, 28(3), p. 245–292.
605. Hacking I. (1975) *The Emergence of Probability*. Cambridge University Press, Cambridge, UK.
606. Hald A. (1990) *A History of Probability and Statistics and Their Applications before 1750*. Wiley, New York.
607. Halpern J. Y. (1990) An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3), p. 311–350.

608. Hamming R. W. (1991) *The Art of Probability for Scientists and Engineers*. Addison-Wesley, Reading, Massachusetts.
609. Hammond K. (1989) *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, New York.
610. Hamscher W., Console L., and Kleer J. D. (1992) *Readings in Model-based Diagnosis*. Morgan Kaufmann, San Mateo, California.
611. Handschin J. E. and Mayne D. Q. (1969) Monte Carlo techniques to estimate the conditional expectation in multi-stage nonlinear filtering. *International Journal of Control*, 9(5), p. 547–559.
612. Hansen E. (1998) Solving POMDPs by searching in policy space. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, p. 211–219, Madison, Wisconsin. Morgan Kaufmann.
613. Hansen E. and Zilberstein S. (2001) LAO\*: a heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1–2), p. 35–62.
614. Hansen P. and Jaumard B. (1990) Algorithms for the maximum satisfiability problem. *Computing*, 44(4), p. 279–303.
615. Hanski I. and Cambefort Y. (Eds.) (1991) *Dung Beetle Ecology*. Princeton University Press, Princeton, New Jersey.
616. Hansson O. and Mayer A. (1989) Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario. Morgan Kaufmann.
617. Hansson O., Mayer A., and Yung M. (1992) Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences*, 63(3), p. 207–227.
618. Haralick R. M. and Elliot G. L. (1980) Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3), p. 263–313.
619. Hardin G. (1968) The tragedy of the commons. *Science*, 162, p. 1243–1248.
620. Harel D. (1984) Dynamic logic. In Gabbay D. and Guenther F. (Eds.), *Handbook of Philosophical Logic*, Vol. 2, p. 497–604. D. Reidel, Dordrecht, Netherlands.
621. Harman G. H. (1983) *Change in View: Principles of Reasoning*. MIT Press, Cambridge, Massachusetts.
622. Harsanyi J. (1967) Games with incomplete information played by Bayesian players. *Management Science*, 14, p. 159–182.
623. Hart P. E., Nilsson N. J., and Raphael B. (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2), p. 100–107.
624. Hart P. E., Nilsson N. J., and Raphael B. (1972) Correction to “A formal basis for the heuristic determination of minimum cost paths”. *SIGART Newsletter*, 37, p. 28–29.
625. Hart T. P. and Edwards D. J. (1961) The tree prune (TP) algorithm. Artificial intelligence project memo 30, Massachusetts Institute of Technology, Cambridge, Massachusetts.
626. Hartley R. and Zisserman A. (2000) *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, UK.
627. Haslum P. and Geffner H. (2001) Heuristic planning with time and resources. In *Proceedings of the IJCAI-01 Workshop on Planning with Resources*, Seattle.
628. Hastie T. and Tibshirani R. (1996) Discriminant adaptive nearest neighbor classification and regression. In Touretzky D. S., Mozer M. C., and Hasselmo M. E. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 8, p. 409–15. MIT Press, Cambridge, Massachusetts.
629. Hastie T., Tibshirani R., and Friedman J. (2001) *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer-Verlag, Berlin.
630. Haugeland J. (Ed.) (1981) *Mind Design*. MIT Press, Cambridge, Massachusetts.
631. Haugeland J. (Ed.) (1985) *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, Massachusetts.
632. Haussler D. (1989) Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1), p. 7–40.

633. Havelund K., Lowry M., Park S., Pecheur C., Penix J., Visser W., and White J. L. (2000) Formal analysis of the remote agent before and after flight. In *Proceedings of the 5th NASA Langley Formal Methods Workshop*, Williamsburg, VA.
634. Hayes P. J. (1978) The naive physics manifesto. In Michie D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.
635. Hayes P. J. (1979) The logic of frames. In Metzing D. (Ed.), *Frame Conceptions and Text Understanding*, p. 46–61. de Gruyter, Berlin.
636. Hayes P. J. (1985a) Naive physics I: Ontology for liquids. In Hobbs J. R. and Moore R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 3, p. 71–107. Ablex, Norwood, New Jersey.
637. Hayes P. J. (1985b) The second naive physics manifesto. In Hobbs J. R. and Moore R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 1, p. 1–36. Ablex, Norwood, New Jersey.
638. Hebb D. O. (1949) *The Organization of Behavior*. Wiley, New York.
639. Heckerman D. (1986) Probabilistic interpretation for MYCIN's certainty factors. In Kanal L. N. and Lemmer J. F. (Eds.), *Uncertainty in Artificial Intelligence*, p. 167–196. Elsevier/North-Holland, Amsterdam, London, New York.
640. Heckerman D. (1991) *Probabilistic Similarity Networks*. MIT Press, Cambridge, Massachusetts.
641. Heckerman D. (1998) A tutorial on learning with Bayesian networks. In Jordan M. I. (Ed.), *Learning in graphical models*. Kluwer, Dordrecht, Netherlands.
642. Heckerman D., Geiger D., and Chickering D. M. (1994) Learning Bayesian networks: The combination of knowledge and statistical data. Technical report MSR-TR-94-09, Microsoft Research, Redmond, Washington.
643. Heim I. and Kratzer A. (1998) *Semantics in a Generative Grammar*. Blackwell, Oxford, UK.
644. Heinz E. A. (2000) *Scalable search in computer chess*. Vieweg, Braunschweig, Germany.
645. Held M. and Karp R. M. (1970) The traveling salesman problem and minimum spanning trees. *Operations Research*, 18, p. 1138–1162.
646. Helmert M. (2001) On the complexity of planning in transportation domains. In Cesta A. and Barrajo D. (Eds.), *Sixth European Conference on Planning (ECP-01)*, Toledo, Spain. Springer-Verlag.
647. Hendrix G. G. (1975) Expanding the utility of semantic networks through partitioning. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, p. 115–121, Tbilisi, Georgia. IJCAII.
648. Henrion M. (1988) Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer J. F. and Kanal L. N. (Eds.), *Uncertainty in Artificial Intelligence 2*, p. 149–163. Elsevier/North-Holland, Amsterdam, London, New York.
649. Henzinger T. A. and Sastry S. (Eds.) (1998) *Hybrid systems: Computation and control*. Springer-Verlag, Berlin.
650. Herbrand J. (1930) *Recherches sur la Théorie de la Démonstration*. Ph.D. thesis, University of Paris.
651. Hewitt C. (1969) PLANNER: a language for proving theorems in robots. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, p. 295–301, Washington, DC. IJCAII.
652. Hierholzer C. (1873) Über die Möglichkeit einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6, p. 30–32.
653. Hilgard E. R. and Bower G. H. (1975) *Theories of Learning* (4th edition). Prentice-Hall, Upper Saddle River, New Jersey.
654. Hintikka J. (1962) *Knowledge and Belief*. Cornell University Press, Ithaca, New York.
655. Hinton G. E. and Anderson J. A. (1981) *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates, Potomac, Maryland.
656. Hinton G. E. and Nowlan S. J. (1987) How learning can guide evolution. *Complex Systems*, 1(3), p. 495–502.

657. Hinton G. E. and Sejnowski T. (1983) Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p. 448–453, Washington, DC. IEEE Computer Society Press.
658. Hinton G. E. and Sejnowski T. (1986) Learning and relearning in Boltzmann machines. In Rumelhart D. E. and McClelland J. L. (Eds.), *Parallel Distributed Processing*, chap. 7, p. 282–317. MIT Press, Cambridge, Massachusetts.
659. Hirsh H. (1987) Explanation-based generalization in a logic programming environment. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan. Morgan Kaufmann.
660. Hirst G. (1981) *Anaphora in Natural Language Understanding: A Survey*, Vol. 119 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin.
661. Hirst G. (1987) *Semantic Interpretation against Ambiguity*. Cambridge University Press, Cambridge, UK.
662. Hobbs J. R. (1978) Resolving pronoun references. *Lingua*, 44, p. 339–352.
663. Hobbs J. R. (1990) *Literature and Cognition*. CSLI Press, Stanford, California.
664. Hobbs J. R., Appelt D., Bear J., Israel D., Kameyama M., Stickel M. E., and Tyson M. (1997) FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Roche E. and Schabes Y. (Eds.), *Finite-State Devices for Natural Language Processing*, p. 383–406. MIT Press, Cambridge, Massachusetts.
665. Hobbs J. R. and Moore R. C. (Eds.) (1985) *Formal Theories of the Commonsense World*. Ablex, Norwood, New Jersey.
666. Hobbs J. R., Stickel M. E., Appelt D., and Martin P. (1993) Interpretation as abduction. *Artificial Intelligence*, 63(1–2), p. 69–142.
667. Hoffmann J. (2000) A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems*, p. 216–227, Charlotte, North Carolina. Springer-Verlag.
668. Hogan N. (1985) Impedance control: An approach to manipulation. Parts i, ii, and iii. *Transactions ASME Journal of Dynamics, Systems, Measurement, and Control*, 107(3), p. 1–24.
669. Holland J. H. (1975) *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan.
670. Holland J. H. (1995) *Hidden order: How adaptation builds complexity*. Addison-Wesley, Reading, Massachusetts.
671. Holldobler S. and Schneeberger J. (1990) A new deductive approach to planning. *New Generation Computing*, 8(3), p. 225–244.
672. Holzmann G. J. (1997) The Spin model checker. *ISSS Transactions on Software Engineering*, 23(5), p. 279–295.
673. Hood A. (1824) Case 4th — 28 July 1824 (Mr. Hood's cases of injuries of the brain). *The Phrenological Journal and Miscellany*, 2, p. 82–94.
674. Hopfield J. J. (1982) Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 79, p. 2554–2558.
675. Horn A. (1951) On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16, p. 14–21.
676. Horn B. K. P. (1970) Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report 232, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.
677. Horn B. K. P. (1986) *Robot Vision*. MIT Press, Cambridge, Massachusetts.
678. Horn B. K. P. and Brooks M. J. (1989) *Shape from Shading*. MIT Press, Cambridge, Massachusetts.
679. Horning J. J. (1969) *A study of grammatical inference*. Ph.D. thesis, Stanford University.

680. Horowitz E. and Sahni S. (1978) *Fundamentals of computer algorithms*. Computer Science Press, Rockville, Maryland.
681. Horswill I. (2000) Functional programming of behavior-based systems. *Autonomous Robots*, 9, p. 83–93.
682. Horvitz E. J. (1987) Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proceedings of the Second Annual NASA Research Forum*, p. 26–43, Moffett Field, California. NASA Ames Research Center.
683. Horvitz E. J. (1989) Rational metareasoning and compilation for optimizing decisions under bounded resources. In *Proceedings of Computational Intelligence 89*, Milan. Association for Computing Machinery.
684. Horvitz E. J. and Barry M. (1995) Display of information for time-critical decision making. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, p. 296–305, Montreal, Canada. Morgan Kaufmann.
685. Horvitz E. J., Breese J. S., Heckerman D., and Hovel D. (1998) The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, p. 256–265, Madison, Wisconsin. Morgan Kaufmann.
686. Horvitz E. J., Breese J. S., and Henrion M. (1988) Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2, p. 247–302.
687. Horvitz E. J. and Breese J. S. (1996) Ideal partition of resources for metareasoning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, p. 1229–1234, Portland, Oregon. AAAI Press.
688. Horvitz E. J. and Heckerman D. (1986) The inconsistent use of measures of certainty in artificial intelligence research. In Kanal L. N. and Lemmer J. F. (Eds.), *Uncertainty in Artificial Intelligence*, p. 137–151. Elsevier/North-Holland, Amsterdam, London, New York.
689. Horvitz E. J., Heckerman D., and Langlotz C. P. (1986) A framework for comparing alternative formalisms for plausible reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, p. 210–214, Philadelphia. Morgan Kaufmann.
690. Hovy E. (1988) *Generating Natural Language under Pragmatic Constraints*. Lawrence Erlbaum, Potomac, Maryland.
691. Howard R. A. (1960) *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts.
692. Howard R. A. (1966) Information value theory. *IEEE Transactions on Systems Science and Cybernetics, SSC-2*, p. 22–26.
693. Howard R. A. (1977) Risk preference. In Howard R. A. and Matheson J. E. (Eds.), *Readings in Decision Analysis*, p. 429–465. Decision Analysis Group, SRI International, Menlo Park, California.
694. Howard R. A. (1989) Microrisks for medical decision analysis. *International Journal of Technology Assessment in Health Care*, 5, p. 357–370.
695. Howard R. A. and Matheson J. E. (1984) Influence diagrams. In Howard R. A. and Matheson J. E. (Eds.) *Readings on the Principles and Applications of Decision Analysis*, p. 721–762. Strategic Decisions Group, Menlo Park, California.
696. Hsu F.-H. (1999) IBM's Deep Blue chess grandmaster chips. *IEEE Micro*, 19(2), p. 70–80.
697. Hsu F.-H., Anantharaman T. S., Campbell M. S., and Nowatzyk A. (1990) A grandmaster chess machine. *Scientific American*, 263(4), p. 44–50.
698. Huang T., Koller D., Malik J., Ogasawara G., Rao B., Russell S. J., and Weber J. (1994) Automatic symbolic traffic scene analysis using belief networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 966–972, Seattle. AAAI Press.
699. Huang X. D., Acero A., and Hon H. (2001) *Spoken Language Processing*. Prentice Hall, Upper Saddle River, New Jersey.
700. Hubel D. H. (1988) *Eye, Brain, and Vision*. W. H. Freeman, New York.

701. Huddleston R. D. and Pullum G. K. (2002) *The Cambridge Grammar of the English Language*. Cambridge University Press, Cambridge, UK.
702. Huffman D. A. (1971) Impossible objects as nonsense sentences. In Meltzer B. and Michie D. (Eds.), *Machine Intelligence 6*, p. 295–324. Edinburgh University Press, Edinburgh, Scotland.
703. Hughes B. D. (1995) *Random Walks and Random Environments, Vol. 1: Random Walks*. Oxford University Press, Oxford, UK.
704. Huhns M. N. and Singh M. P. (Eds.) (1998) *Readings in agents*. Morgan Kaufmann, San Mateo, California.
705. Hume D. (1739) *A Treatise of Human Nature* (2nd edition). Republished by Oxford University Press, 1978, Oxford, UK.
706. Hunsberger L. and Grosz B. J. (2000) A combinatorial auction for collaborative planning. In *International Conference on Multi-Agent Systems (ICMAS-2000)*.
707. Hunt E. B., Marin J., and Stone P. T. (1966) *Experiments in Induction*. Academic Press, New York.
708. Hunter L. and States D. J. (1992) Bayesian classification of protein structure. *IEEE Expert*, 7(4), p. 67–75.
709. Hurwicz L. (1973) The design of mechanisms for resource allocation. *American Economic Review Papers and Proceedings*, 63(1), p. 1–30.
710. Hutchins W. J. and Somers H. (1992) *An Introduction to Machine Translation*. Academic Press, New York.
711. Huttenlocher D. P. and Ullman S. (1990) Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2), p. 195–212.
712. Huygens C. (1657) Ratiociniis in ludo aleae. In van Schooten F. (Ed.), *Exercitionum Mathematicorum*. Elsevirii, Amsterdam.
713. Hwa R. (1998) An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of COLING-ACL '98*, p. 557–563, Montreal. International Committee on Computational Linguistics and Association for Computational Linguistics.
714. Hwang C. H. and Schubert L. K. (1993) EL: A formal, yet natural, comprehensive knowledge representation. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, p. 676–682, Washington, DC. AAAI Press.
715. Indyk P. (2000) Dimensionality reduction techniques for proximity problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, p. 371–378, San Francisco. Association for Computing Machinery.
716. Ingberman P. Z. (1967) Panini-Backus form suggested. *Communications of the Association for Computing Machinery*, 10(3), p. 137.
717. Inoue K. (2001) Inverse entailment for full clausal theories. In *LICS-2001 Workshop on Logic and Learning*, Boston. IEEE.
718. Intille S. and Bobick A. (1999) A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, p. 518–525, Orlando, Florida. AAAI Press.
719. Isard M. and Blake A. (1996) Contour tracking by stochastic propagation of conditional density. In *Proceedings of Fourth European Conference on Computer Vision*, p. 343–356, Cambridge, UK. Springer-Verlag.
720. Jaakkola T. and Jordan M. I. (1996) Computing upper and lower bounds on likelihoods in intractable networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, p. 340–348. Morgan Kaufmann, Portland, Oregon.
721. Jaakkola T., Singh S. P., and Jordan M. I. (1995) Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro G., Touretzky D., and Leen T. (Eds.) *Advances in Neural Information Processing Systems 7*, p. 345–352, Cambridge, Massachusetts. MIT Press.

722. Jaffar J. and Lassez J.-L. (1987) Constraint logic programming. In *Proceedings of the Fourteenth ACM Conference on Principles of Programming Languages*, p. 111–119, Munich. Association for Computing Machinery.
723. Jaffar J., Michaylov S., Stuckey P. J., and Yap R. H. C. (1992a) The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3), p. 339–395.
724. Jaffar J., Stuckey P. J., Michaylov S., and Yap R. H. C. (1992b) An abstract machine for CLP(R). *SIGPLAN Notices*, 27(7), p. 128–139.
725. Jaskowski S. (1934) On the rules of suppositions in formal logic. *Studia Logica*, 1.
726. Jefferson G. (1949) The mind of mechanical man: The Lister Oration delivered at the Royal College of Surgeons in England. *British Medical Journal*, 1(25), p. 1105–1121.
727. Jeffrey R. C. (1983) *The Logic of Decision* (2nd edition). University of Chicago Press, Chicago.
728. Jeffreys H. (1948) *Theory of Probability*. Oxford, Oxford, UK.
729. Jelinek F. (1969) Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 64, p. 532–556.
730. Jelinek F. (1976) Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4), p. 532–556.
731. Jelinek F. (1997) *Statistical methods for speech recognition*. MIT Press, Cambridge, Massachusetts.
732. Jelinek F. and Mercer R. L. (1980) Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, p. 381–397, Amsterdam, London, New York. North Holland.
733. Jennings H. S. (1906) *Behavior of the lower organisms*. Columbia University Press, New York.
734. Jensen F. V. (2001) *Bayesian Networks and Decision Graphs*. Springer-Verlag, Berlin.
735. Jespersen O. (1965) *Essentials of English Grammar*. University of Alabama Press, Tuscaloosa, Alabama.
736. Jeavons W. S. (1874) *The Principles of Science*. Routledge/Thoemmes Press, London.
737. Jimenez P. and Torras C. (2000) An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artificial Intelligence*, 124(1), p. 1–30.
738. Joachims T. (2001) A statistical learning model of text classification with support vector machines. In *Proceedings of the 24th Conference on Research and Development in Information Retrieval (SIGIR)*, p. 128–136, New Orleans. Association for Computing Machinery.
739. Johnson W. W. and Story W. E. (1879) Notes on the “15” puzzle. *American Journal of Mathematics*, 2, p. 397–404.
740. Johnson-Laird P. N. (1988) *The Computer and the Mind: An Introduction to Cognitive Science*. Harvard University Press, Cambridge, Massachusetts.
741. Johnston M. D. and Adorf H.-M. (1992) Scheduling with neural networks: The case of the Hubble space telescope. *Computers & Operations Research*, 19(3–4), p. 209–240.
742. Jones N. D., Gomard C. K., and Sestoft P. (1993) *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Upper Saddle River, New Jersey.
743. Jones R., Laird J. E., and Nielsen P. E. (1998) Automated intelligent pilots for combat flight simulation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 1047–54, Madison, Wisconsin. AAAI Press.
744. Jonsson A., Morris P., Muscettola N., Rajan K., and Smith B. (2000) Planning in interplanetary space: Theory and practice. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, p. 177–186, Breckenridge, Colorado. AAAI Press.
745. Jordan M. I. (1995) Why the logistic function? a tutorial discussion on probabilities and neural networks. Computational cognitive science technical report 9503, Massachusetts Institute of Technology.
746. Jordan M. I. (2003) *An Introduction to Graphical Models*. В печати.

747. Jordan M. I., Ghahramani Z., Jaakkola T., and Saul L. K. (1998) An introduction to variational methods for graphical models. In Jordan M. I. (Ed.) *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands.
748. Jordan M. I., Ghahramani Z., Jaakkola T., and Saul L. K. (1999) An introduction to variational methods for graphical models. *Machine Learning*, 37(2-3), p. 183–233.
749. Joshi A. K. (1985) Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions. In Dowty D., Karttunen L., and Zwicky A. (Eds.) *Natural Language Parsing*. Cambridge University Press, Cambridge, UK.
750. Joshi A. K., Webber B. L., and Sag I. (1981) *Elements of Discourse Understanding*. Cambridge University Press, Cambridge, UK.
751. Joslin D. and Pollack M. E. (1994) Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1506, Seattle. AAAI Press.
752. Jouannaud J.-P. and Kirchner C. (1991) Solving equations in abstract algebras: A rule-based survey of unification. In Lassez J.-L. and Plotkin G. (Eds.), *Computational Logic*, p. 257–321. MIT Press, Cambridge, Massachusetts.
753. Judd J. S. (1990) *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, Massachusetts.
754. Juels A. and Wattenberg M. (1996) Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. In Touretzky D. S., Mozer M. C., and Hasselmo M. E. (Eds.) *Advances in Neural Information Processing Systems*, Vol. 8, p. 430–436. MIT Press, Cambridge, Massachusetts.
755. Julesz B. (1971) *Foundations of Cyclopean Perception*. University of Chicago Press, Chicago.
756. Jurafsky D. and Martin J. H. (2000) *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, Upper Saddle River, New Jersey.
757. Kadane J. B. and Larkey P. D. (1982) Subjective probability and the theory of games. *Management Science*, 28(2), p. 113–120.
758. Kaelbling L. P., Littman M. L., and Cassandra A. R. (1998) Planning and action in partially observable stochastic domains. *Artificial Intelligence*, 101, p. 99–134.
759. Kaelbling L. P., Littman M. L., and Moore A. W. (1996) Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, p. 237–285.
760. Kaelbling L. P. and Rosenschein S. J. (1990) Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6(1–2), p. 35–48.
761. Kager R. (1999) *Optimality Theory*. Cambridge University Press, Cambridge, UK.
762. Kahneman D., Slovic P., and Tversky A. (Eds.) (1982) *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge, UK.
763. Kaindl H. and Khorsand A. (1994) Memory-bounded bidirectional search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1359–1364, Seattle. AAAI Press.
764. Kalman R. (1960) A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82, p. 35–46.
765. Kambhampati S. (1994) Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, 10, p. 213–244.
766. Kambhampati S., Mali A. D., and Srivastava B. (1998) Hybrid planning for partially hierarchical domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 882–888, Madison, Wisconsin. AAAI Press.
767. Kanal L. N. and Kumar V. (1988) *Search in Artificial Intelligence*. Springer-Verlag, Berlin.
768. Kanal L. N. and Lemmer J. F. (Eds.) (1986) *Uncertainty in Artificial Intelligence*. Elsevier/North-Holland, Amsterdam, London, New York.

769. Kanazawa K., Koller D., and Russell S. J. (1995) Stochastic simulation algorithms for dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, p. 346–351, Montreal, Canada. Morgan Kaufmann.
770. Kaplan D. and Montague R. (1960) A paradox regained. *Notre Dame Journal of Formal Logic*, 1(3), p. 79–90.
771. Karmarkar N. (1984) A new polynomial-time algorithm for linear programming. *Combinatorica*, 4, p. 373–395.
772. Karp R. M. (1972) Reducibility among combinatorial problems. In Miller R. E. and Thatcher J. W. (Eds.) *Complexity of Computer Computations*, p. 85–103. Plenum, New York.
773. Kasami T. (1965) An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.
774. Kasparov G. (1997) IBM owes me a rematch. *Time*, 149(21), p. 66, 67.
775. Kasper R. T. (1988) Systemic grammar and functional unification grammar. In Benson J. and Greaves W. (Eds.) *Systemic Functional Approaches to Discourse*. Ablex, Norwood, New Jersey.
776. Kaufmann M., Manolios P., and Moore J. S. (2000) *Computer-Aided Reasoning: An Approach*. Kluwer, Dordrecht, Netherlands.
777. Kautz H., McAllester D. A., and Selman B. (1996) Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, p. 374–384, Cambridge, Massachusetts. Morgan Kaufmann.
778. Kautz H. and Selman B. (1992) Planning as satisfiability. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, p. 359–363, Vienna. Wiley.
779. Kautz H. and Selman B. (1998) BLACKBOX: A new approach to the application of theorem proving to problem solving. Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search.
780. Kavraki L., Svestka P., Latombe J.-C., and Overmars M. (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), p. 566–580.
781. Kay M., Gawron J. M., and Norvig P. (1994) *Verbmobil: A Translation System for Face-To-Face Dialog*. CSLI Press, Stanford, California.
782. Kaye R. (2000) Minesweeper is NP-complete! *Mathematical Intelligencer*, 5(22), p. 9–15.
783. Kearns M. (1990) *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts.
784. Kearns M., Mansour Y., and Ng A. Y. (2000) Approximate planning in large POMDPs via reusable trajectories. In Solla S. A., Leen T. K., and Müller K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge, Massachusetts.
785. Kearns M. and Singh S. P. (1998) Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning*, p. 260–268, Madison, Wisconsin. Morgan Kaufmann.
786. Kearns M. and Vazirani U. (1994) *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts.
787. Keeney R. L. (1974) Multiplicative utility functions. *Operations Research*, 22, p. 22–34.
788. Keeney R. L. and Raiffa H. (1976) *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, New York.
789. Kehler A. (1997) Probabilistic coreference in information extraction. In Cardie C. and Weischedel R. (Eds.), *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, p. 163–173. Association for Computational Linguistics, Somerset, New Jersey.
790. Kemp M. (Ed.) (1989) *Leonardo on Painting: An Anthology of Writings*. Yale University Press, New Haven, Connecticut.

791. Kern C. and Greenstreet M. R. (1999) Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 4(2), p. 123–193.
792. Keynes J. M. (1921) *A Treatise on Probability*. Macmillan, London.
793. Khatib O. (1986) Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, 5(1), p. 90–98.
794. Kietz J.-U. and Dzeroski S. (1994) Inductive logic programming and learnability. *SIGART Bulletin*, 5(1), p. 22–32.
795. Kim J. H. (1983) *CONVINCE: A Conversational Inference Consolidation Engine*. Ph.D. thesis, Department of Computer Science, University of California at Los Angeles.
796. Kim J. H. and Pearl J. (1983) A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, p. 190–193, Karlsruhe, Germany. Morgan Kaufmann.
797. Kim J. H. and Pearl J. (1987) CONVINCE: A conversational inference consolidation engine. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(2), p. 120–132.
798. King R. D., Muggleton S. H., Lewis R. A., and Sternberg M. J. E. (1992) Drug design by machine learning: The use of inductive logic programming to model the structure activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences of the United States of America*, 89(23), p. 11322–11326.
799. Kirkpatrick S., Gelatt C. D., and Vecchi M. P. (1983) Optimization by simulated annealing. *Science*, 220, p. 671–680.
800. Kirkpatrick S. and Selman B. (1994) Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264(5163), p. 1297–1301.
801. Kirousis L. M. and Papadimitriou C. H. (1988) The complexity of recognizing polyhedral scenes. *Journal of Computer and System Sciences*, 37(1), 14–38.
802. Kitano H., Asada M., Kuniyoshi Y., Noda I., and Osawa E. (1997) RoboCup: The robot world cup initiative. In Johnson W. L. and Hayes-Roth B. (Eds.) *Proceedings of the First International Conference on Autonomous Agents*, p. 340–347, New York. ACM Press.
803. Kjaerulff U. (1992) A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference*, p. 121–129, Stanford, California. Morgan Kaufmann.
804. Klein D. and Manning C. D. (2001) Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *Proceedings of the 39th Annual Meeting of the ACL*.
805. Kleinberg J. M. (1999) Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), p. 604–632.
806. Knight K. (1999) A statistical mt tutorial workbook. prepared in connection with the Johns Hopkins University summer workshop.
807. Knoblock C. A. (1990) Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, p. 923–928, Boston. MIT Press.
808. Knuth D. E. (1968) Semantics for context-free languages. *Mathematical Systems Theory*, 2(2), p. 127–145.
809. Knuth D. E. (1973) *The Art of Computer Programming* (second edition). Vol. 2: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts<sup>1</sup>.
810. Knuth D. E. (1975) An analysis of alpha–beta pruning. *Artificial Intelligence*, 6(4), p. 293–326.
811. Knuth D. E. and Bendix P. B. (1970) Simple word problems in universal algebras. In Leech J. (Ed.), *Computational Problems in Abstract Algebra*, p. 263–267. Pergamon, Oxford, UK.

<sup>1</sup> Кнут, Дональд Э. Искусство программирования, том 1. Основные алгоритмы. 3-е издание: Пер. с англ. — М.: Издательский дом "Вильямс", 2000.

812. Koditschek D. (1987) Exact robot navigation by means of potential functions: some topological considerations. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Vol. 1, p. 1–6, Raleigh, North Carolina. IEEE Computer Society Press.
813. Koehler J., Nebel B., Hoffman J., and Dimopoulos Y. (1997) Extending planning graphs to an ADL subset. In *Proceedings of the Fourth European Conference on Planning*, p. 273–285, Toulouse, France. Springer-Verlag.
814. Koenderink J. J. (1990) *Solid Shape*. MIT Press, Cambridge, Massachusetts.
815. Koenderink J. J. and van Doorn A. J. (1975) Invariant properties of the motion parallax field due to the movement of rigid bodies relative to an observer. *Optica Acta*, 22(9), p. 773–791.
816. Koenderink J. J. and van Doorn A. J. (1991) Affine structure from motion. *Journal of the Optical Society of America A*, 8, p. 377–385.
817. Koenig S. (1991) Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Master's report, Computer Science Division, University of California, Berkeley.
818. Koenig S. (2000) Exploring unknown environments with real-time search or reinforcement learning. In Solla S. A., Leen T. K., and Müller K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge, Massachusetts.
819. Koenig S. and Simmons R. (1998) Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *aips98*. AAAI Press, Menlo Park, California.
820. Kohn W. (1991) Declarative control architecture. *Communications of the Association for Computing Machinery*, 34(8), p. 65–79.
821. Koller D., Megiddo N., and von Stengel B. (1996) Efficient computation of equilibria for extensive two-person games. *Games and Economic Behaviour*, 14(2), p. 247–259.
822. Koller D. and Pfeffer A. (1997) Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1–2), p. 167–215.
823. Koller D. and Pfeffer A. (1998) Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 580–587, Madison, Wisconsin. AAAI Press.
824. Koller D. and Sahami M. (1997) Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, p. 170–178. Morgan Kaufmann.
825. Kolmogorov A. N. (1941) Interpolation und extrapolation von stationären zufälligen folgen. *Bulletin of the Academy of Sciences of the USSR, Ser. Math.* 5, p. 3–14.
826. Kolmogorov A. N. (1950) *Foundations of the Theory of Probability*. Chelsea, New York.
827. Kolmogorov A. N. (1963) On tables of random numbers. *Sankhya, the Indian Journal of Statistics, Series A* 25.
828. Kolmogorov A. N. (1965) Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1(1), p. 1–7.
829. Kolodner J. (1983) Reconstructive memory: A computer model. *Cognitive Science*, 7, p. 281–328.
830. Kolodner J. (1993) *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California.
831. Kondrak G. and van Beek P. (1997) A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89, p. 365–387.
832. Konolige K. (1997) COLBERT: A language for reactive control in Saphira. In *KI-97: Advances in Artificial Intelligence*, LNAI, p. 31–52. Springer Verlag.
833. Konolige K. (1982) A first order formalization of knowledge and action for a multi-agent planning system. In Hayes J. E., Michie D., and Pao Y.-H. (Eds.) *Machine Intelligence 10*. Ellis Horwood, Chichester, England.
834. Koopmans T. C. (1972) Representation of preference orderings over time. In McGuire C. B. and Radner R. (Eds.) *Decision and Organization*. Elsevier/North-Holland, Amsterdam, London, New York.

835. Korf R. E. (1985a) Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1), p. 97–109.
836. Korf R. E. (1985b) Iterative-deepening A\*: An optimal admissible tree search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, p. 1034–1036, Los Angeles. Morgan Kaufmann.
837. Korf R. E. (1987) Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1), p. 65–88.
838. Korf R. E. (1988) Optimal path finding algorithms. In Kanal L. N. and Kumar V. (Eds.) *Search in Artificial Intelligence*, chap. 7, p. 223–267. Springer-Verlag, Berlin.
839. Korf R. E. (1990) Real-time heuristic search. *Artificial Intelligence*, 42(3), p. 189–212.
840. Korf R. E. (1991) Best-first search with limited memory. UCLA Computer Science Annual.
841. Korf R. E. (1993) Linear-space best-first search. *Artificial Intelligence*, 62(1), p. 41–78.
842. Korf R. E. (1995) Space-efficient search algorithms. *ACM Computing Surveys*, 27(3), p. 337–339.
843. Korf R. E. and Chickering D. M. (1996) Best-first minimax search. *Artificial Intelligence*, 84(1–2), p. 299–337.
844. Korf R. E. and Felner A. (2002) Disjoint pattern database heuristics. *Artificial Intelligence*, 134(1–2), p. 9–22.
845. Korf R. E. and Zhang W. (2000) Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 17th National Conference on Artificial Intelligence*, p. 910–916, Cambridge, Massachusetts. MIT Press.
846. Kortenkamp D., Bonasso R. P., and Murphy R. (Eds.) (1998) *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA. MIT Press.
847. Kotok A. (1962) A chess playing program for the IBM 7090. Ai project memo 41, MIT Computation Center, Cambridge, Massachusetts.
848. Koutsoupias E. and Papadimitriou C. H. (1992) On the greedy algorithm for satisfiability. *Information Processing Letters*, 43(1), p. 53–55.
849. Kowalski R. (1974) Predicate logic as a programming language. In *Proceedings of the IFIP-74 Congress*, p. 569–574. Elsevier/North-Holland.
850. Kowalski R. (1979a) Algorithm = logic + control. *Communications of the Association for Computing Machinery*, 22, p. 424–436.
851. Kowalski R. (1979b) *Logic for Problem Solving*. Elsevier/North-Holland, Amsterdam, London, New York.
852. Kowalski R. (1988) The early years of logic programming. *Communications of the Association for Computing Machinery*, 31, p. 38–43.
853. Kowalski R. and Kuehner D. (1971) Linear resolution with selection function. *Artificial Intelligence*, 2(3–4), p. 227–260.
854. Kowalski R. and Sergot M. (1986) A logic-based calculus of events. *New Generation Computing*, 4(1), p. 67–95.
855. Koza J. R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts.
856. Koza J. R. (1994) *Genetic Programming II: Automatic discovery of reusable programs*. MIT Press, Cambridge, Massachusetts.
857. Koza J. R., Bennett F. H., Andre D., and Keane M. A. (1999) *Genetic Programming III: Darwinian invention and problem solving*. Morgan Kaufmann, San Mateo, California.
858. Kraus S., Ephrati E., and Lehmann D. (1991) Negotiation in a non-cooperative environment. *Journal of Experimental and Theoretical Artificial Intelligence*, 3(4), 255–281.
859. Kripke S. A. (1963) Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16, p. 83–94.
860. Krovetz R. (1993) Viewing morphology as an inference process. In *Proceedings of the Sixteenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, p. 191–202, New York. ACM Press.

861. Kruppa E. (1913) Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung. *Sitz.-Ber. Akad. Wiss., Wien, Math. Naturw., Kl. Abt. IIa*, 122, p. 1939–1948.
862. Kuhn H. W. (1953) Extensive games and the problem of information. In Kuhn H. W. and Tucker A. W. (Eds.), *Contributions to the Theory of Games II*. Princeton University Press, Princeton, New Jersey.
863. Kuipers B. J. and Levitt T. S. (1988) Navigation and mapping in large-scale space. *AI Magazine*, 9(2), p. 25–43.
864. Kukich K. (1992) Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4), p. 377–439.
865. Kumar P. R. and Varaiya P. (1986) *Stochastic systems: Estimation, identification, and adaptive control*. Prentice-Hall, Upper Saddle River, New Jersey.
866. Kumar V. (1992) Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1), p. 32–44.
867. Kumar V. and Kanal L. N. (1983) A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, 21, p. 179–198.
868. Kumar V. and Kanal L. N. (1988) The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In Kanal L. N. and Kumar V. (Eds.), *Search in Artificial Intelligence*, chap. 1, p. 1–27. Springer-Verlag, Berlin.
869. Kumar V., Nau D. S., and Kanal L. N. (1988) A general branch-and-bound formulation for AND/OR graph and game tree search. In Kanal L. N. and Kumar V. (Eds.), *Search in Artificial Intelligence*, chap. 3, p. 91–130. Springer-Verlag, Berlin.
870. Kuper G. M. and Vardi M. Y. (1993) On the complexity of queries in the logical data model. *Theoretical Computer Science*, 116(1), p. 33–57.
871. Kurzweil R. (1990) *The Age of Intelligent Machines*. MIT Press, Cambridge, Massachusetts.
872. Kurzweil R. (2000) *The Age of Spiritual Machines*. Penguin.
873. Kyburg H. E. (1977) Randomness and the right reference class. *The Journal of Philosophy*, 74(9), p. 501–521.
874. Kyburg H. E. (1983) The reference class. *Philosophy of Science*, 50, p. 374–397.
875. La Mettrie J. O. (1748) *L'homme machine*. E. Luzac, Leyde, France.
876. La Mura P. and Shoham Y. (1999) Expected utility networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, p. 366–373, Stockholm. Morgan Kaufmann.
877. Ladkin P. (1986a) Primitives and units for time specification. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, p. 354–359, Philadelphia. Morgan Kaufmann.
878. Ladkin P. (1986b) Time representation: a taxonomy of interval relations. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, p. 360–366, Philadelphia. Morgan Kaufmann.
879. Lafferty J. and Zhai C. (2001) Probabilistic relevance models based on document and query generation. In *Proceedings of the Workshop on Language Modeling and Information retrieval*.
880. Laird J. E., Newell A., and Rosenbloom P. S. (1987) SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1), p. 1–64.
881. Laird J. E., Rosenbloom P. S., and Newell A. (1986) Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1, p. 11–46.
882. Lakoff G. (1987) *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. University of Chicago Press, Chicago.
883. Lakoff G. and Johnson M. (1980) *Metaphors We Live By*. University of Chicago Press, Chicago.
884. Lamarck J. B. (1809) *Philosophie zoologique*. Chez Dentu et L'Auteur, Paris.

885. Langley P., Simon H. A., Bradshaw G. L., and Zytkow J. M. (1987) *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT Press, Cambridge, Massachusetts.
886. Langton C. (Ed.) (1995) *Artificial life*. MIT Press, Cambridge, Massachusetts.
887. Laplace P. (1816) *Essai philosophique sur les probabilités* (3rd edition). Courcier Imprimeur, Paris.
888. Lappin S. and Leass H. J. (1994) An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4), p. 535–561.
889. Lari K. and Young S. J. (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer, Speech and Language*, 4, p. 35–56.
890. Larrañaga P., Kuijpers C., Murga R., Inza I., and Dizdarevic S. (1999) Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13, p. 129–170.
891. Latombe J.-C. (1991) *Robot Motion Planning*. Kluwer, Dordrecht, Netherlands.
892. Lauritzen S. (1995) The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19, p. 191–201.
893. Lauritzen S. (1996) *Graphical models*. Oxford University Press, Oxford, UK.
894. Lauritzen S., Dawid A., Larsen B., and Leimer H. (1990) Independence properties of directed Markov fields. *Networks*, 20(5), p. 491–505.
895. Lauritzen S. and Spiegelhalter D. J. (1988) Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B* 50(2), p. 157–224.
896. Lauritzen S. and Wermuth N. (1989) Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17, p. 31–57.
897. Lavrač N. and Džeroski S. (1994) *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, England.
898. Lawler E. L. (1985) *The traveling salesman problem: A guided tour of combinatorial optimization*. Wiley, New York.
899. Lawler E. L., Lenstra J. K., Kan A., and Shmoys D. B. (1992) *The Travelling Salesman Problem*. Wiley Interscience.
900. Lawler E. L., Lenstra J. K., Kan A., and Shmoys D. B. (1993) Sequencing and scheduling: algorithms and complexity. In Graves S. C., Zipkin P. H., and Kan A. H. G. R. (Eds.) *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science, Volume 4*, p. 445–522. North-Holland, Amsterdam.
901. Lawler E. L. and Wood D. E. (1966) Branch-and-bound methods: A survey. *Operations Research*, 14(4), p. 699–719.
902. Lazanas A. and Latombe J.-C. (1992) Landmark-based robot navigation. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, p. 816–822, San Jose. AAAI Press.
903. Le Cun Y., Jackel L., Boser B., and Denker J. (1989) Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11), p. 41–46.
904. LeCun Y., Jackel L., Bottou L., Brunot A., Cortes C., Denker J., Drucker H., Guyon I., Muller U., Sackinger E., Simard P., and Vapnik V. N. (1995) Comparison of learning algorithms for handwritten digit recognition. In Fogelman F. and Gallinari P. (Eds.), *International Conference on Artificial Neural Networks*, p. 53–60, Berlin. Springer-Verlag.
905. Leech G., Rayson P., and Wilson A. (2001) *Word Frequencies in Written and Spoken English: Based on the British National Corpus*. Longman, New York.
906. Lefkovitz D. (1960) A strategic pattern recognition program for the game Go. Technical note 60-243, Wright Air Development Division, University of Pennsylvania, Moore School of Electrical Engineering.
907. Lenat D. B. (1983) EURISKO: A program that learns new heuristics and domain concepts: The nature of heuristics, III: Program design and results. *Artificial Intelligence*, 21(1–2), p. 61–98.

908. Lenat D. B. (1995) Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11).
909. Lenat D. B. and Brown J. S. (1984) Why AM and EURISKO appear to work. *Artificial Intelligence*, 23(3), p. 269–294.
910. Lenat D. B. and Guha R. V. (1990) *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts.
911. Leonard H. S. and Goodman N. (1940) The calculus of individuals and its uses. *Journal of Symbolic Logic*, 5(2), p. 45–55.
912. Leonard J. J. and Durrant-Whyte H. (1992) *Directed sonar sensing for mobile robot navigation*. Kluwer, Dordrecht, Netherlands.
913. Leśniewski S. (1916) Podstawy ogólnej teorii mnogości. Moscow.
914. Lettvin J. Y., Maturana H. R., McCulloch W. S., and Pitts W. H. (1959) What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11), p. 1940–1951.
915. Letz R., Schumann J., Bayerl S., and Bibel W. (1992) SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, 8(2), p. 183–212.
916. Levesque H. J. and Brachman R. J. (1987) Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3(2), p. 78–93.
917. Levesque H. J., Reiter R., Lespérance Y., Lin F., and Scherl R. (1997a) GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31, p. 59–84.
918. Levesque H. J., Reiter R., Lespérance Y., Lin F., and Scherl R. (1997b) GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31, p. 59–84.
919. Levitt G. M. (2000) *The Turk, Chess Automaton*. McFarland and Company.
920. Levy D. N. L. (Ed.) (1988a) *Computer Chess Compendium*. Springer-Verlag, Berlin.
921. Levy D. N. L. (Ed.) (1988b) *Computer Games*. Springer-Verlag, Berlin.
922. Lewis D. D. (1998) Naive Bayes at forty: The independence assumption in information retrieval. In *Machine Learning: ECML-98. 10th European Conference on Machine Learning. Proceedings*, p. 4–15, Chemnitz, Germany. Springer-Verlag.
923. Lewis D. K. (1966) An argument for the identity theory. *The Journal of Philosophy*, 63(1), p. 17–25.
924. Lewis D. K. (1972) General semantics. In Davidson D. and Harman G. (Eds.), *Semantics of Natural Language*, p. 169–218. D. Reidel, Dordrecht, Netherlands.
925. Lewis D. K. (1980) Mad pain and Martian pain. In Block N. (Ed.), *Readings in Philosophy of Psychology*, Vol. 1, p. 216–222. Harvard University Press, Cambridge, Massachusetts.
926. Li C. M. and Anbulagan (1997) Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, p. 366–371, Nagoya, Japan. Morgan Kaufmann.
927. Li M. and Vitanyi P. M. B. (1993) *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin.
928. Lifschitz V. (1986) On the semantics of STRIPS. In Georgeff M. P. and Lansky A. L. (Eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, p. 1–9, Timberline, Oregon. Morgan Kaufmann.
929. Lifschitz V. (2001) Answer set programming and plan generation. *Artificial Intelligence*, 138(1–2), p. 39–54.
930. Lighthill J. (1973) Artificial intelligence: A general survey. In Lighthill J., Sutherland N. S., Needham R. M., Longuet-Higgins H. C., and Michie D. (Eds.), *Artificial Intelligence: A Paper Symposium*. Science Research Council of Great Britain, London.
931. Lin F. and Reiter R. (1997) How to progress a database. *Artificial Intelligence*, 92(1–2), p. 131–167.
932. Lin S. (1965) Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, 44(10), 2245–2269.

933. Lin S. and Kernighan B. W. (1973) An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, 21(2), p. 498–516.
934. Linden T. A. (1991) Representing software designs as partially developed plans. In Lowry M. R. and McCartney R. D. (Eds.), *Automating Software Design*, p. 603–625. MIT Press, Cambridge, Massachusetts.
935. Lindsay R. K. (1963) Inferential memory as the basis of machines which understand natural language. In Feigenbaum E. A. and Feldman J. (Eds.), *Computers and Thought*, p. 217–236. McGraw-Hill, New York.
936. Lindsay R. K., Buchanan B. G., Feigenbaum E. A., and Lederberg J. (1980) *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. McGraw-Hill, New York.
937. Littman M. L. (1994) Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, p. 157–163, New Brunswick, NJ. Morgan Kaufmann.
938. Littman M. L., Keim G. A., and Shazeer N. M. (1999) Solving crosswords with PROVERB. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, p. 914–915, Orlando, Florida. AAAI Press.
939. Liu J. S. and Chen R. (1998) Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93, p. 1022–1031.
940. Lloyd J. W. (1987) *Foundations of Logic Programming*. Springer-Verlag, Berlin.
941. Locke J. (1690) *An Essay Concerning Human Understanding*. William Tegg.
942. Locke W. N. and Booth A. D. (1955) *Machine Translation of Languages: Fourteen Essays*. MIT Press, Cambridge, Massachusetts.
943. Lodge D. (1984) *Small World*. Penguin Books, New York.
944. Lohn J. D., Kraus W. F., and Colombano S. P. (2001) Evolutionary optimization of yagi-uda antennas. In *Proceedings of the Fourth International Conference on Evolvable Systems*, p. 236–243.
945. Longuet-Higgins H. C. (1981) A computer algorithm for reconstructing a scene from two projections. *Nature*, 293, p. 133–135.
946. Lovejoy W. S. (1991) A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1–4), p. 47–66.
947. Loveland D. (1968) Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery*, 15(2), p. 236–251.
948. Loveland D. (1970) A linear format for resolution. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, p. 147–162, Berlin. Springer-Verlag.
949. Loveland D. (1984) Automated theorem-proving: A quarter-century review. *Contemporary Mathematics*, 29, p. 1–45.
950. Lowe D. G. (1987) Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31, p. 355–395.
951. Löwenheim L. (1915) Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76, p. 447–470.
952. Lowerre B. T. (1976) *The HARPY Speech Recognition System*. Ph.D. thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.
953. Lowerre B. T. and Reddy R. (1980) The HARPY speech recognition system. In Lea W. A. (Ed.), *Trends in Speech Recognition*, chap. 15. Prentice-Hall, Upper Saddle River, New Jersey.
954. Lowry M. R. and McCartney R. D. (1991) *Automating Software Design*. MIT Press, Cambridge, Massachusetts.
955. Loyd S. (1959) *Mathematical Puzzles of Sam Loyd: Selected and Edited by Martin Gardner*. Dover, New York.
956. Lozano-Perez T. (1983) Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2), p. 108–120.

957. Lozano-Perez T., Mason M., and Taylor R. (1984) Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), p. 3–24.
958. Luby M., Sinclair A., and Zuckerman D. (1993) Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47, p. 173–180.
959. Luby M. and Vigoda E. (1999) Fast convergence of the glauber dynamics for sampling independent sets. *Random Structures and Algorithms*, 15(3–4), p. 229–241.
960. Lucas J. R. (1961) Minds, machines, and Gödel. *Philosophy*, 36.
961. Lucas J. R. (1976) This Gödel is killing me: A rejoinder. *Philosophia*, 6(1), p. 145–148.
962. Lucas P. (1996) Knowledge acquisition for decision-theoretic expert systems. *AISB Quarterly*, 94, p. 23–33.
963. Luce D. R. and Raiffa H. (1957) *Games and Decisions*. Wiley, New York.
964. Luger G. F. (Ed.) (1995) *Computation and intelligence: Collected readings*. AAAI Press, Menlo Park, California.
965. MacKay D. J. C. (1992) A practical Bayesian framework for back-propagation networks. *Neural Computation*, 4(3), p. 448–472.
966. Mackworth A. K. (1973) Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, 4, p. 121–137.
967. Mackworth A. K. (1977) Consistency in networks of relations. *Artificial Intelligence*, 8(1), p. 99–118.
968. Mackworth A. K. (1992) Constraint satisfaction. In Shapiro S. (Ed.) *Encyclopedia of Artificial Intelligence* (second edition), Vol. 1, p. 285–293. Wiley, New York.
969. Mahanti A. and Daniels C. J. (1993) A SIMD approach to parallel heuristic search. *Artificial Intelligence*, 60(2), p. 243–282.
970. Majercik S. M. and Littman M. L. (1999) Planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, p. 549–556.
971. Malik J. (1987) Interpreting line drawings of curved objects. *International Journal of Computer Vision*, 1(1), p. 73–103.
972. Malik J. and Rosenholtz R. (1994) Recovering surface curvature and orientation from texture distortion: A least squares algorithm and sensitivity analysis. In Eklundh J.-O. (Ed.) *Proceedings of the Third European Conf. on Computer Vision*, p. 353–364, Stockholm. Springer-Verlag.
973. Malik J. and Rosenholtz R. (1997) Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision*, 23(2), p. 149–168.
974. Mann W. C. and Thompson S. A. (1983) Relational propositions in discourse. Tech. rep. RR-83-115, Information Sciences Institute.
975. Mann W. C. and Thompson S. A. (1988) Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3), p. 243–281.
976. Manna Z. and Waldinger R. (1971) Toward automatic program synthesis. *Communications of the Association for Computing Machinery*, 14(3), p. 151–165.
977. Manna Z. and Waldinger R. (1985) *The Logical Basis for Computer Programming: Volume 1: Deductive Reasoning*. Addison-Wesley, Reading, Massachusetts.
978. Manna Z. and Waldinger R. (1986) Special relations in automated deduction. *Journal of the Association for Computing Machinery*, 33(1), p. 1–59.
979. Manna Z. and Waldinger R. (1992) Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, 18(8), p. 674–704.
980. Manning C. D. and Schütze H. (1999) *Foundations of Statistical Natural Language Processing*. MIT Press.
981. Marbach P. and Tsitsiklis J. N. (1998) Simulation-based optimization of Markov reward processes. Technical report LIDS-P-2411, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

982. Marcus M. P., Santorini B., and Marcinkiewicz M. A. (1993) Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2), p. 313–330.
983. Markov A. A. (1913) An example of statistical investigation in the text of “Eugene Onegin” illustrating coupling of “tests” in chains. *Proceedings of the Academy of Sciences of St. Petersburg*, 7.
984. Maron M. E. (1961) Automatic indexing: An experimental inquiry. *Journal of the Association for Computing Machinery*, 8(3), p. 404–417.
985. Maron M. E. and Kuhns J.-L. (1960) On relevance, probabilistic indexing and information retrieval. *Communications of the ACM*, 7, p. 219–244.
986. Marr D. (1982) *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, New York.
987. Marriott K. and Stuckey P. J. (1998) *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Massachusetts.
988. Marsland A. T. and Schaeffer J. (Eds.) (1990) *Computers, Chess, and Cognition*. Springer-Verlag, Berlin.
989. Martelli A. and Montanari U. (1976) Unification in linear time and space: A structured presentation. Internal report B 76-16, Istituto di Elaborazione della Informazione, Pisa, Italy.
990. Martelli A. and Montanari U. (1978) Optimizing decision trees through heuristically guided search. *Communications of the Association for Computing Machinery*, 21, p. 1025–1039.
991. Marthi B., Pasula H., Russell S. J., and Peres Y. (2002) Decayed MCMC filtering. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference*, p. 319–326, Edmonton, Alberta. Morgan Kaufmann.
992. Martin J. H. (1990) *A Computational Model of Metaphor Interpretation*. Academic Press, New York.
993. Martin P. and Shmoys D. B. (1996) A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of the 5th International IPCO Conference*, p. 389–403. Springer-Verlag.
994. Maslov S. Y. (1964) An inverse method for establishing deducibility in classical predicate calculus. *Doklady Akademii nauk SSSR*, 159, p. 17–20.
995. Maslov S. Y. (1967) An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. *Doklady Akademii nauk SSSR*, 172, p. 22–25.
996. Mason M. (1993) Kicking the sensing habit. *AI Magazine*, 14(1), p. 58–59.
997. Mason M. (2001) *Mechanics of Robotic Manipulation*. MIT Press.
998. Mason M. and Salisbury J. (1985) *Robot hands and the mechanics of manipulation*. MIT Press.
999. Mataric M. J. (1997) Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1), p. 73–83.
1000. Mates B. (1953) *Stoic Logic*. University of California Press, Berkeley and Los Angeles.
1001. Maxwell J. and Kaplan R. (1993) The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4), p. 571–590.
1002. Maxwell J. and Kaplan R. (1995) A method for disjunctive constraint satisfaction. In Dalrymple M., Kaplan R., Maxwell J., and Zaenen A. (Eds.) *Formal Issues in Lexical-Functional Grammar*, No. 47 in CSLI Lecture Note Series, chap. 14, p. 381–481. CSLI Publications.
1003. McAllester D. A. (1980) An outlook on truth maintenance. Ai memo 551, MIT AI Laboratory, Cambridge, Massachusetts.
1004. McAllester D. A. (1988) Conspiracy numbers for min-max search. *Artificial Intelligence*, 35(3), p. 287–310.
1005. McAllester D. A. (1989) *Ontic: A Knowledge Representation System for Mathematics*. MIT Press, Cambridge, Massachusetts.
1006. McAllester D. A. (1998) What is the most pressing issue facing ai and the aaai today? Candidate statement, election for Councilor of the American Association for Artificial Intelligence.

1007. McAllester D. A. and Givan R. (1992) Natural language syntax and first-order inference. *Artificial Intelligence*, 56(1), p. 1–20.
1008. McAllester D. A. and Rosenblitt D. (1991) Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Vol. 2, p. 634–639, Anaheim, California. AAAI Press.
1009. McCarthy J. (1958) Programs with common sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, Vol. 1, p. 77–84, London. Her Majesty's Stationery Office.
1010. McCarthy J. (1963) Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project, Stanford, California.
1011. McCarthy J. (1968) Programs with common sense. In Minsky M. L. (Ed.) *Semantic Information Processing*, p. 403–418. MIT Press, Cambridge, Massachusetts.
1012. McCarthy J. (1980) Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2), p. 27–39.
1013. McCarthy J. and Hayes P. J. (1969) Some philosophical problems from the standpoint of artificial intelligence. In Meltzer B., Michie D., and Swann M. (Eds.) *Machine Intelligence 4*, p. 463–502. Edinburgh University Press, Edinburgh, Scotland.
1014. McCarthy J., Minsky M. L., Rochester N., and Shannon C. E. (1955) Proposal for the Dartmouth summer research project on artificial intelligence. Tech. rep., Dartmouth College.
1015. McCawley J. D. (1988) *The Syntactic Phenomena of English*, Vol. 2 volumes. University of Chicago Press.
1016. McCawley J. D. (1993) *Everything That Linguists Have Always Wanted to Know about Logic but Were Ashamed to Ask* (Second edition). University of Chicago Press, Chicago.
1017. McCulloch W. S. and Pitts W. (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, p. 115–137.
1018. McCune W. (1992) Automated discovery of new axiomatizations of the left group and right group calculi. *Journal of Automated Reasoning*, 9(1), p. 1–24.
1019. McCune W. (1997) Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3), p. 263–276.
1020. McDermott D. (1976) Artificial intelligence meets natural stupidity. *SIGART Newsletter*, 57, p. 4–9.
1021. McDermott D. (1978a) Planning and acting. *Cognitive Science*, 2(2), p. 71–109.
1022. McDermott D. (1978b) Tarskian semantics, or, no notation without denotation! *Cognitive Science*, 2(3).
1023. McDermott D. (1987) A critique of pure reason. *Computational Intelligence*, 3(3), p. 151–237.
1024. McDermott D. (1996) A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on AI Planning Systems*, p. 142–149, Edinburgh, Scotland. AAAI Press.
1025. McDermott D. and Doyle J. (1980) Non-monotonic logic: i. *Artificial Intelligence*, 13(1–2), p. 41–72.
1026. McDermott D. (1982) R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19(1), p. 39–88.
1027. McEliece R. J., MacKay D. J. C., and Cheng J.-F. (1998) Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2), p. 140–152.
1028. McGregor J. J. (1979) Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19(3), p. 229–250.
1029. McKeown K. (1985) *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, Cambridge, UK.
1030. McLachlan G. J. and Krishnan T. (1997) *The EM Algorithm and Extensions*. Wiley, New York.
1031. McMillan K. L. (1993) *Symbolic Model Checking*. Kluwer, Dordrecht, Netherlands.
1032. Meehl P. (1955) *Clinical vs. Statistical Prediction*. University of Minnesota Press, Minneapolis.
1033. Melčuk I. A. and Polguere A. (1988) A formal lexicon in the meaning-text theory (or how to do lexica with words) *Computational Linguistics*, 13(3–4), p. 261–275.

1034. Mendel G. (1866) Versuche über pflanzen-hybriden. *Verhandlungen des Naturforschenden Vereins, Abhandlungen, Brünn*, 4, p. 3–47. Translated into English by C. T. Druery, published by Bateson (1902).
1035. Mercer J. (1909) Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London, A*, 209, p. 415–446.
1036. Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., and Teller E. (1953) Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, p. 1087–1091.
1037. Mézard M. and Nadal J.-P. (1989) Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics*, 22, p. 2191–2204.
1038. Michalski R. S. (1969) On the quasi-minimal solution of the general covering problem. In *Proceedings of the First International Symposium on Information Processing*, p. 125–128.
1039. Michalski R. S., Carbonell J. G., and Mitchell T. M. (Eds.) (1983) *Machine Learning: An Artificial Intelligence Approach*, Vol. 1. Morgan Kaufmann, San Mateo, California.
1040. Michalski R. S., Carbonell J. G., and Mitchell T. M. (Eds.) (1986a) *Machine Learning: An Artificial Intelligence Approach*, Vol. 2. Morgan Kaufmann, San Mateo, California.
1041. Michalski R. S., Mozetic I., Hong J., and Lavrač, N. (1986b) The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, p. 1041–1045, Philadelphia. Morgan Kaufmann.
1042. Michel S. and Plamondon P. (1996) Bilingual sentence alignment: Balancing robustness and accuracy. In *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA)*.
1043. Michie D. (1966) Game-playing and game-learning automata. In Fox L. (Ed.), *Advances in Programming and Non-Numerical Computation*, p. 183–200. Pergamon, Oxford, UK.
1044. Michie D. (1972) Machine intelligence at Edinburgh. *Management Informatics*, 2(1), p. 7–12.
1045. Michie D. (1974) Machine intelligence at Edinburgh. In *On Intelligence*, p. 143–155. Edinburgh University Press.
1046. Michie D. and Chambers R. A. (1968) BOXES: An experiment in adaptive control. In Dale E. and Michie D. (Eds.), *Machine Intelligence 2*, p. 125–133. Elsevier/North-Holland, Amsterdam, London, New York.
1047. Michie D., Spiegelhalter D. J., and Taylor C. (Eds.) (1994) *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Chichester, England.
1048. Milgrom P. (1997) Putting auction theory to work: The simultaneous ascending auction. Tech. rep. Technical Report 98-0002, Stanford University Department of Economics.
1049. Mill J. S. (1843) *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence, and Methods of Scientific Investigation*. J. W. Parker, London.
1050. Mill J. S. (1863) *Utilitarianism*. Parker, Son and Bourn, London.
1051. Miller A. C., Merkhofer M. M., Howard R. A., Matheson J. E., and Rice T. R. (1976) Development of automated aids for decision analysis. Technical report, SRI International, Menlo Park, California.
1052. Minsky M. L. (Ed.) (1968) *Semantic Information Processing*. MIT Press, Cambridge, Massachusetts.
1053. Minsky M. L. (1975) A framework for representing knowledge. In Winston P. H. (Ed.), *The Psychology of Computer Vision*, p. 211–277. McGraw-Hill, New York. Originally an MIT AI Laboratory memo; the 1975 version is abridged, but is the most widely cited.
1054. Minsky M. L. and Papert S. (1969) *Perceptrons: An Introduction to Computational Geometry* (first edition). MIT Press, Cambridge, Massachusetts.
1055. Minsky M. L. and Papert S. (1988) *Perceptrons: An Introduction to Computational Geometry* (Expanded edition). MIT Press, Cambridge, Massachusetts.
1056. Minton S. (1984) Constraint-based generalization: Learning game-playing plans from single examples. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, p. 251–254, Austin, Texas. Morgan Kaufmann.

1057. Minton S. (1988) Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, p. 564–569, St. Paul, Minnesota. Morgan Kaufmann.
1058. Minton S., Johnston M. D., Philips A. B., and Laird P. (1992) Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1–3), p. 161–205.
1059. Mitchell M. (1996) *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts.
1060. Mitchell M., Holland J. H., and Forrest S. (1996) When will a genetic algorithm outperform hill climbing? In Cowan J., Tesauro G., and Alspector J. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 6. MIT Press, Cambridge, Massachusetts.
1061. Mitchell T. M. (1977) Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, p. 305–310, Cambridge, Massachusetts. IJCAII.
1062. Mitchell T. M. (1982) Generalization as search. *Artificial Intelligence*, 18(2), p. 203–226.
1063. Mitchell T. M. (1990) Becoming increasingly reactive (mobile robots). In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, p. 1051–1058, Boston. MIT Press.
1064. Mitchell T. M. (1997) *Machine Learning*. McGraw-Hill, New York.
1065. Mitchell T. M., Keller R., and Kedar-Cabelli S. (1986) Explanation-based generalization: A unifying view. *Machine Learning*, 1, p. 47–80.
1066. Mitchell T. M., Utgoff P. E., and Banerji R. (1983) Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski R. S., Carbonell J. G., and Mitchell T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, p. 163–190. Morgan Kaufmann, San Mateo, California.
1067. Mitkov R. (2002) *Anaphora Resolution*. Longman, New York.
1068. Mohr R. and Henderson T. C. (1986) Arc and path consistency revisited. *Artificial Intelligence*, 28(2), p. 225–233.
1069. Mohri M., Pereira F., and Riley M. (2002) Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1), p. 69–88.
1070. Montague P. R., Dayan P., Person C., and Sejnowski T. (1995) Bee foraging in uncertain environments using predictive Hebbian learning. *Nature*, 377, p. 725–728.
1071. Montague R. (1970) English as a formal language. In *Linguaggi nella Società e nella Tecnica*, p. 189–224. Edizioni di Comunità, Milan.
1072. Montague R. (1973) The proper treatment of quantification in ordinary English. In Hintikka K. J. J., Moravcsik J. M. E., and Suppes P. (Eds.) *Approaches to Natural Language*. D. Reidel, Dordrecht, Netherlands.
1073. Montanari U. (1974) Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(2), p. 95–132.
1074. Montemerlo M., Thrun S., Koller D., and Wegbreit B. (2002) FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, Edmonton, Alberta. AAAI Press.
1075. Mooney R. (1999) Learning for semantic interpretation: Scaling up without dumbing down. In Cussens J. (Ed.), *Proceedings of the 1st Workshop on Learning Language in Logic*, p. 7–15. Springer-Verlag.
1076. Mooney R. J. and Califf M. E. (1995) Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of AI Research*, 3, p. 1–24.
1077. Moore A. W. and Atkeson C. G. (1993) Prioritized sweeping — reinforcement learning with less data and less time. *Machine Learning*, 13, p. 103–130.
1078. Moore E. F. (1959) The shortest path through a maze. In *Proceedings of an International Symposium on the Theory of Switching, Part II*, p. 285–292. Harvard University Press, Cambridge, Massachusetts.

1079. Moore J. S. and Newell A. (1973) How can Merlin understand? In Gregg L. (Ed.), *Knowledge and Cognition*. Lawrence Erlbaum Associates, Potomac, Maryland.
1080. Moore R. C. (1980) Reasoning about knowledge and action. Artificial intelligence center technical note 191, SRI International, Menlo Park, California.
1081. Moore R. C. (1985) A formal theory of knowledge and action. In Hobbs J. R. and Moore R. C. (Eds.), *Formal Theories of the Commonsense World*, p. 319–358. Ablex, Norwood, New Jersey.
1082. Moravec H. P. (1983) The stanford cart and the cmu rover. *Proceedings of the IEEE*, 71(7), p. 872–884.
1083. Moravec H. P. and Elfes A. (1985) High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, p. 116–121, St. Louis, Missouri. IEEE Computer Society Press.
1084. Moravec H. P. (1988) *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, Cambridge, Massachusetts.
1085. Moravec H. P. (2000) *Robot: Mere Machine to Transcendent Mind*. Oxford University Press.
1086. Morgenstern L. (1987) Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, p. 867–874, Milan. Morgan Kaufmann.
1087. Morgenstern L. (1998) Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *Artificial Intelligence*, 103, p. 237–271.
1088. Morjaria M. A., Rink F. J., Smith W. D., Klempner G., Burns C., and Stein J. (1995) Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, p. 141–148. Morgan Kaufmann.
1089. Morrison P. and Morrison E. (Eds.) (1961) *Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others*. Dover, New York.
1090. Moskewicz M. W., Madigan C. F., Zhao Y., Zhang L., and Malik S. (2001) Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, p. 530–535. ACM Press.
1091. Mosteller F. and Wallace D. L. (1964) *Inference and Disputed Authorship: The Federalist*. Addison-Wesley.
1092. Mostow J. and Prieditis A. E. (1989) Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 1, p. 701–707, Detroit. Morgan Kaufmann.
1093. Motzkin T. S. and Schoenberg I. J. (1954) The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3), p. 393–404.
1094. Moussouris J., Holloway J., and Greenblatt R. D. (1979) CHEOPS: A chess-oriented processing system. In Hayes J. E., Michie D., and Mikulich L. I. (Eds.), *Machine Intelligence 9*, p. 351–360. Ellis Horwood, Chichester, England.
1095. Moutarlier P. and Chatila R. (1989) Stochastic multisensory data fusion for mobile robot location and environment modeling. In *5th Int. Symposium on Robotics Research*, Tokyo.
1096. Muggleton S. H. (1991) Inductive logic programming. *New Generation Computing*, 8, p. 295–318.
1097. Muggleton S. H. (1992) *Inductive Logic Programming*. Academic Press, New York.
1098. Muggleton S. H. (1995) Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4), p. 245–286.
1099. Muggleton S. H. (2000) Learning stochastic logic programs. *Proceedings of the AAAI 2000 Workshop on Learning Statistical Models from Relational Data*.
1100. Muggleton S. H. and Buntine W. (1988) Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, p. 339–352. Morgan Kaufmann.

1101. Muggleton S. H. and De Raedt L. (1994) Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20, p. 629–679.
1102. Muggleton S. H. and Feng C. (1990) Efficient induction of logic programs. In *Proceedings of the Workshop on Algorithmic Learning Theory*, p. 368–381, Tokyo. Ohmsha.
1103. Müller M. (2002) Computer Go. *Artificial Intelligence*, 134(1–2), p. 145–179.
1104. Mundy J. and Zisserman A. (Eds.) (1992) *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, Massachusetts.
1105. Murphy K., Weiss Y., and Jordan M. I. (1999) Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, p. 467–475, Stockholm. Morgan Kaufmann.
1106. Murphy K. and Russell S. J. (2001) Rao-blackwellised particle filtering for dynamic bayesian networks. In Doucet A., de Freitas N., and Gordon N. J. (Eds.) *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, Berlin.
1107. Murphy R. (2000) *Introduction to AI Robotics*. MIT Press, Cambridge, Massachusetts.
1108. Muscettola N., Nayak P., Pell B., and Williams B. (1998) Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103, p. 5–48.
1109. Myerson R. B. (1991) *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge.
1110. Nagel T. (1974) What is it like to be a bat? *Philosophical Review*, 83, p. 435–450.
1111. Nalwa V. S. (1993) *A Guided Tour of Computer Vision*. Addison-Wesley, Reading, Massachusetts.
1112. Nash J. (1950) Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36, p. 48–49.
1113. Nau D. S. (1980) Pathology on game trees: A summary of results. In *Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80)*, p. 102–104, Stanford, California. AAAI.
1114. Nau D. S. (1983) Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence*, 21(1–2), p. 221–244.
1115. Nau D. S., Kumar V., and Kanal L. N. (1984) General branch and bound, and its relation to A\* and AO\*. *Artificial Intelligence*, 23, p. 29–58.
1116. Naur P. (1963) Revised report on the algorithmic language Algol 60. *Communications of the Association for Computing Machinery*, 6(1), p. 1–17.
1117. Nayak P. and Williams B. (1997) Fast context switching in real-time propositional reasoning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, p. 50–56, Providence, Rhode Island. AAAI Press.
1118. Neal R. (1996) *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin.
1119. Nebel B. (2000) On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research*, 12, p. 271–315.
1120. Nelson G. and Oppen D. C. (1979) Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2), p. 245–257.
1121. Netto E. (1901) *Lehrbuch der Combinatorik*. B. G. Teubner, Leipzig.
1122. Nevill-Manning C. G. and Witten I. H. (1997) Identifying hierarchical structures in sequences: A linear-time algorithm. *Journal of AI Research*, 7, p. 67–82.
1123. Nevins A. J. (1975) Plane geometry theorem proving using forward chaining. *Artificial Intelligence*, 6(1), p. 1–23.
1124. Newell A. (1982) The knowledge level. *Artificial Intelligence*, 18(1), p. 82–127.
1125. Newell A. (1990) *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts.
1126. Newell A. and Ernst G. (1965) The search for generality. In Kalenich W. A. (Ed.), *Information Processing 1965: Proceedings of IFIP Congress 1965*, Vol. 1, p. 17–24, Chicago. Spartan.

1127. Newell A., Shaw J. C., and Simon H. A. (1957) Empirical explorations with the logic theory machine. *Proceedings of the Western Joint Computer Conference*, 15, p. 218–239. Перепечатано в [459].
1128. Newell A., Shaw J. C., and Simon H. A. (1958) Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, 4(2), p. 320–335.
1129. Newell A. and Simon H. A. (1961) GPS, a program that simulates human thought. In Billing H. (Ed.) *Lernende Automaten*, p. 109–124. R. Oldenbourg, Munich.
1130. Newell A. and Simon H. A. (1972) *Human Problem Solving*. Prentice-Hall, Upper Saddle River, New Jersey.
1131. Newell A. and Simon H. A. (1976) Computer science as empirical inquiry: Symbols and search. *Communications of the Association for Computing Machinery*, 19, p. 113–126.
1132. Newton I. (1664–1671) Methodus fluxionum et serierum infinitarum. Неопубликованные заметки.
1133. Ng A. Y., Harada D., and Russell S. J. (1999) Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia. Morgan Kaufmann.
1134. Ng A. Y. and Jordan M. I. (2000) PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference*, p. 406–415, Stanford, California. Morgan Kaufmann.
1135. Nguyen X. and Kambhampati S. (2001) Reviving partial order planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, p. 459–466, Seattle. Morgan Kaufmann.
1136. Nguyen X., Kambhampati S., and Nigenda R. S. (2001) Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. Tech. rep., Computer Science and Engineering Department, Arizona State University.
1137. Nicholson A. and Brady J. M. (1992) The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, p. 689–693, Vienna, Austria. Wiley.
1138. Niemelä, I., Simons P., and Syrjänen T. (2000) Smodels: A system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*.
1139. Nilsson D. and Lauritzen S. (2000) Evaluating influence diagrams using LIMIDs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference*, p. 436–445, Stanford, California. Morgan Kaufmann.
1140. Nilsson N. J. (1965) *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, New York. republished in 1990.
1141. Nilsson N. J. (1971) *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York.
1142. Nilsson N. J. (1980) *Principles of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.
1143. Nilsson N. J. (1984) Shakey the robot. Technical note 323, SRI International, Menlo Park, California.
1144. Nilsson N. J. (1986) Probabilistic logic. *Artificial Intelligence*, 28(1), p. 71–87.
1145. Nilsson N. J. (1991) Logic and artificial intelligence. *Artificial Intelligence*, 47(1–3), p. 31–56.
1146. Nilsson N. J. (1998) *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Mateo, California.
1147. Norvig P. (1988) Multiple simultaneous interpretations of ambiguous sentences. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*.
1148. Norvig P. (1992) *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, San Mateo, California.
1149. Nowick S. M., Dean M. E., Dill D. L., and Horowitz M. (1993) The design of a high-performance cache controller: A case study in asynchronous synthesis. *Integration: The VLSI Journal*, 15(3), p. 241–262.

1150. Nunberg G. (1979) The non-uniqueness of semantic solutions: Polysemy. *Language and Philosophy*, 3(2), p. 143–184.
1151. Nussbaum M. C. (1978) *Aristotle's "De Motu Animalium"*. Princeton University Press, Princeton, New Jersey.
1152. Ogawa S., Lee T.-M., Kay A. R., and Tank D. W. (1990) Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of the National Academy of Sciences of the United States of America*, 87, p. 9868–9872.
1153. Olawsky D. and Gini M. (1990) Deferred planning and sensor use. In Sycara K. P. (Ed.), *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, California. Defense Advanced Research Projects Agency (DARPA), Morgan Kaufmann.
1154. Olesen K. G. (1993) Causal probabilistic networks with both discrete and continuous variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 15(3), p. 275–279.
1155. Oliver R. M. and Smith J. Q. (Eds.) (1990) *Influence Diagrams, Belief Nets and Decision Analysis*. Wiley, New York.
1156. Olson C. F. (1994) Time and space efficient pose clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 251–258, Washington, DC. IEEE Computer Society Press.
1157. Oncina J. and Garcia P. (1992) Inferring regular languages in polynomial update time. In Perez, Sanfeliu, and Vidal (Eds.) *Pattern Recognition and Image Analysis*, p. 49–61. World Scientific.
1158. O'Reilly U.-M. and Oppacher F. (1994) Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Davidor Y., Schwefel H.-P., and Manner R. (Eds.) *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, p. 397–406, Jerusalem, Israel. Springer-Verlag.
1159. Ormoneit D. and Sen S. (2002) Kernel-based reinforcement learning. *Machine Learning*, 49(2–3), p. 161–178.
1160. Ortony A. (Ed.) (1979) *Metaphor and Thought*. Cambridge University Press, Cambridge, UK.
1161. Osborne M. J. and Rubinstein A. (1994) *A Course in Game Theory*. MIT Press, Cambridge, Massachusetts.
1162. Osherson D. N., Stob M., and Weinstein S. (1986) *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, Massachusetts.
1163. Page C. D. and Srinivasan A. (2002) ILP: A short look back and a longer look forward. Submitted to Journal of Machine Learning Research.
1164. Pak I. (2001) On mixing of certain random walks, cutoff phenomenon and sharp threshold of random matroid processes. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 110, p. 251–272.
1165. Palay A. J. (1985) *Searching with Probabilities*. Pitman, London.
1166. Palmer D. A. and Hearst M. A. (1994) Adaptive sentence boundary disambiguation. In *Proceedings of the Conference on Applied Natural Language Processing*, p. 78–83. Morgan Kaufmann.
1167. Palmer S. (1999) *Vision Science: Photons to Phenomenology*. MIT Press, Cambridge, Massachusetts.
1168. Papadimitriou C. H. (1994) *Computational Complexity*. Addison Wesley.
1169. Papadimitriou C. H., Tamaki H., Raghavan P., and Vempala S. (1998) Latent semantic indexing: A probabilistic analysis. In *Proceedings of the ACM Conference on Principles of Database Systems (PODS)*, p. 159–168, New York. ACM Press.
1170. Papadimitriou C. H. and Tsitsiklis J. N. (1987) The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3), p. 441–450.
1171. Papadimitriou C. H. and Yannakakis M. (1991) Shortest paths without a map. *Theoretical Computer Science*, 84(1), p. 127–150.

1172. Papavassiliou V. and Russell S. J. (1999) Convergence of reinforcement learning with general function approximators. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, p. 748–757, Stockholm. Morgan Kaufmann.
1173. Parekh R. and Honavar V. (2001) Dfa learning from simple examples. *Machine Learning*, 44, p. 9–35.
1174. Parisi G. (1988) *Statistical field theory*. Addison-Wesley, Reading, Massachusetts.
1175. Parker D. B. (1985) Learning logic. Technical report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.
1176. Parker L. E. (1996) On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 10(6).
1177. Parr R. and Russell S. J. (1998) Reinforcement learning with hierarchies of machines. In Jordan M. I., Kearns M., and Solla S. A. (Eds.), *Advances in Neural Information Processing Systems 10*. MIT Press, Cambridge, Massachusetts.
1178. Parzen E. (1962) On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33, p. 1065–1076.
1179. Pasula H. and Russell S. J. (2001) Approximate inference for first-order probabilistic languages. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle. Morgan Kaufmann.
1180. Pasula H., Russell S. J., Ostland M., and Ritov Y. (1999) Tracking many objects with many sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm. Morgan Kaufmann.
1181. Paterson M. S. and Wegman M. N. (1978) Linear unification. *Journal of Computer and System Sciences*, 16, p. 158–167.
1182. Patrick B. G., Almulla M., and Newborn M. M. (1992) An upper bound on the time complexity of iterative-deepening-A\*. *Annals of Mathematics and Artificial Intelligence*, 5(2–4), p. 265–278.
1183. Patten T. (1988) *Systemic Text Generation as Problem Solving*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK.
1184. Paul R. P. (1981) *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Massachusetts.
1185. Peano G. (1889) *Arithmetices principia, nova methodo exposita*. Fratres Bocca, Turin.
1186. Pearl J. (1982a) Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, p. 133–136, Pittsburgh, Pennsylvania. Morgan Kaufmann.
1187. Pearl J. (1982b) The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the Association for Computing Machinery*, 25(8), p. 559–564.
1188. Pearl J. (1984) *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusetts.
1189. Pearl J. (1986) Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29, p. 241–288.
1190. Pearl J. (1987) Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32, p. 247–257.
1191. Pearl J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.
1192. Pearl J. (2000) *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, UK.
1193. Pearl J. and Verma T. (1991) A theory of inferred causation. In Allen J. A., Fikes R., and Sandewall E. (Eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, p. 441–452, San Mateo, California. Morgan Kaufmann.
1194. Pearson J. and Jeavons P. (1997) A survey of tractable constraint satisfaction problems. Technical report CSD-TR-97-15, Royal Holloway College U. of London.

1195. Pednault E. P. D. (1986) Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff M. P. and Lansky A. L. (Eds.) *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, p. 47–82, Timberline, Oregon. Morgan Kaufmann.
1196. Peirce C. S. (1870) Description of a notation for the logic of relatives, resulting from an amplification of the conceptions of Boole's calculus of logic. *Memoirs of the American Academy of Arts and Sciences*, 9, p. 317–378.
1197. Peirce C. S. (1883) A theory of probable inference. Note B. The logic of relatives. In *Studies in Logic by Members of the Johns Hopkins University*, p. 187–203, Boston.
1198. Peirce C. S. (1902) Logic as semiotic: The theory of signs. Неопубликованная рукопись; перепечатана в Buchler B. (1955) *Philosophical Writings of Peirce*. Dover Publications, New York.
1199. Peirce C. S. (1909) Existential graphs. Неопубликованная рукопись; перепечатана в Buchler B. (1955) *Philosophical Writings of Peirce*. Dover Publications, New York.
1200. Pelikan M., Goldberg D. E., and Cantu-Paz E. (1999) BOA: The Bayesian optimization algorithm. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, p. 525–532, Orlando, Florida. Morgan Kaufmann.
1201. Pemberton J. C. and Korf R. E. (1992) Incremental planning on graphs with cycles. In Hendler J. (Ed.), *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, p. 525–532, College Park, Maryland. Morgan Kaufmann.
1202. Penberthy J. S. and Weld D. S. (1992) UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, p. 103–114. Morgan Kaufmann.
1203. Peng J. and Williams R. J. (1993) Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 2, p. 437–454.
1204. Penrose R. (1989) *The Emperor's New Mind*. Oxford University Press, Oxford, UK.
1205. Penrose R. (1994) *Shadows of the Mind*. Oxford University Press, Oxford, UK.
1206. Peot M. and Smith D. E. (1992) Conditional nonlinear planning. In Hendler J. (Ed.), *Proceedings of the First International Conference on AI Planning Systems*, p. 189–197, College Park, Maryland. Morgan Kaufmann.
1207. Pereira F. and Shieber S. M. (1987) *Prolog and Natural-Language Analysis*. Center for the Study of Language and Information (CSLI), Stanford, California.
1208. Pereira F. and Warren D. H. D. (1980) Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13, p. 231–278.
1209. Peterson C. and Anderson J. R. (1987) A mean field theory learning algorithm for neural networks. *Complex Systems*, 1(5), p. 995–1019.
1210. Pfeffer A. (2000) *Probabilistic Reasoning for Complex Systems*. Ph.D. thesis, Stanford University, Stanford, California.
1211. Pinker S. (1989) *Learnability and Cognition*. MIT Press, Cambridge, MA.
1212. Pinker S. (1995) Language acquisition. In Gleitman L. R., Liberman M., and Osherson D. N. (Eds.), *An Invitation to Cognitive Science* (second edition), Vol. 1. MIT Press, Cambridge, Massachusetts.
1213. Pinker S. (2000) *The Language Instinct: How the Mind Creates Language*. MIT Press, Cambridge, Massachusetts.
1214. Plaat A., Schaeffer J., Pijls W., and de Bruin A. (1996) Best-first fixed-depth minimax algorithms. *Artificial Intelligence Journal*, 87(1–2), p. 255–293.
1215. Place U. T. (1956) Is consciousness a brain process? *British Journal of Psychology*, 47, p. 44–50.
1216. Plotkin G. (1971) *Automatic Methods of Inductive Inference*. Ph.D. thesis, Edinburgh University.
1217. Plotkin G. (1972) Building-in equational theories. In Meltzer B. and Michie D. (Eds.), *Machine Intelligence* 7, p. 73–90. Edinburgh University Press, Edinburgh, Scotland.

1218. Pnueli A. (1977) The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, p. 46–57, Providence, Rhode Island. IEEE, IEEE Computer Society Press.
1219. Pohl I. (1969) Bi-directional and heuristic search in path problems. Tech. rep. 104, SLAC (Stanford Linear Accelerator Center), Stanford, California.
1220. Pohl I. (1970) First results on the effect of error in heuristic search. In Meltzer B. and Michie D. (Eds.), *Machine Intelligence 5*, p. 219–236. Elsevier/North-Holland, Amsterdam, London, New York.
1221. Pohl I. (1971) Bi-directional search. In Meltzer B. and Michie D. (Eds.) *Machine Intelligence 6*, p. 127–140. Edinburgh University Press, Edinburgh, Scotland.
1222. Pohl I. (1973) The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, p. 20–23, Stanford, California. IJCAI.
1223. Pohl I. (1977) Practical and theoretical considerations in heuristic search algorithms. In Elcock E. W. and Michie D. (Eds.), *Machine Intelligence 8*, p. 55–72. Ellis Horwood, Chichester, England.
1224. Pomerleau D. A. (1993) *Neural Network Perception for Mobile Robot Guidance*. Kluwer, Dordrecht, Netherlands.
1225. Ponte J. M. and Croft W. B. (1998) A language modeling approach to information retrieval. In *Research and Development in Information Retrieval*, p. 275–281.
1226. Poole D. (1993) Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64, p. 81–129.
1227. Poole D., Mackworth A. K., and Goebel R. (1998) *Computational intelligence: A logical approach*. Oxford University Press, Oxford, UK.
1228. Popper K. R. (1959) *The Logic of Scientific Discovery*. Basic Books, New York.
1229. Popper K. R. (1962) *Conjectures and Refutations: The Growth of Scientific Knowledge*. Basic Books, New York.
1230. Porter M. F. (1980) An algorithm for suffix stripping. *Program*, 13(3), p. 130–137.
1231. Post E. L. (1921) Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43, p. 163–185.
1232. Pradhan M., Provan G. M., Middleton B., and Henrion M. (1994) Knowledge engineering for large belief networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, p. 484–490, Seattle, Washington. Morgan Kaufmann.
1233. Pratt V. R. (1976) Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Symposium on the Foundations of Computer Science*, p. 109–121. IEEE Computer Society Press.
1234. Prawitz D. (1960) An improved proof procedure. *Theoria*, 26, 102–139.
1235. Prawitz D. (1965) *Natural Deduction: A Proof Theoretical Study*. Almqvist and Wiksell, Stockholm.
1236. Press W. H., Teukolsky S. A., Vetterling W. T., and Flannery B. P. (2002) *Numerical Recipes in C++: The Art of Scientific Computing* (Second edition). Cambridge University Press, Cambridge, UK.
1237. Prieditis A. E. (1993) Machine discovery of effective admissible heuristics. *Machine Learning*, 12(1–3), p. 117–141.
1238. Prinz D. G. (1952) Robot chess. *Research*, 5, p. 261–266.
1239. Prior A. N. (1967) *Past, Present, and Future*. Oxford University Press, Oxford, UK.
1240. Prosser P. (1993) Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9, p. 268–299.
1241. Pryor L. and Collins G. (1996) Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4, p. 287–339.
1242. Pullum G. K. (1991) *The Great Eskimo Vocabulary Hoax (and Other Irreverent Essays on the Study of Language)*. University of Chicago Press, Chicago.

1243. Pullum G. K. (1996) Learnability, hyperlearning, and the poverty of the stimulus. In *22nd Annual Meeting of the Berkeley Linguistics Society*.
1244. Puterman M. L. (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York.
1245. Puterman M. L. and Shin M. C. (1978) Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11), p. 1127–1137.
1246. Putnam H. (1960) Minds and machines. In Hook S. (Ed.), *Dimensions of Mind*, p. 138–164. Macmillan, London.
1247. Putnam H. (1963) “Degree of confirmation” and inductive logic. In Schilpp P. A. (Ed.) *The Philosophy of Rudolf Carnap*, p. 270–292. Open Court, La Salle, Illinois.
1248. Putnam H. (1967) The nature of mental states. In Capitan W. H. and Merrill D. D. (Eds.) *Art, Mind, and Religion*, p. 37–48. University of Pittsburgh Press, Pittsburgh.
1249. Pylyshyn Z. W. (1974) Minds, machines and phenomenology: Some reflections on Dreyfus’ “What Computers Can’t Do”. *International Journal of Cognitive Psychology*, 3(1), p. 57–77.
1250. Pylyshyn Z. W. (1984) *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press, Cambridge, Massachusetts.
1251. Quillian M. R. (1961) A design for an understanding machine. Paper presented at a colloquium: Semantic Problems in Natural Language, King’s College, Cambridge, England.
1252. Quine W. V. (1953) Two dogmas of empiricism. In *From a Logical Point of View*, p. 20–46. Harper and Row, New York.
1253. Quine W. V. (1960) *Word and Object*. MIT Press, Cambridge, Massachusetts.
1254. Quine W. V. (1982) *Methods of Logic* (Fourth edition). Harvard University Press, Cambridge, Massachusetts.
1255. Quinlan E. and O’Brien S. (1992) Sublanguage: Characteristics and selection guidelines for MT. In *AI and Cognitive Science ’92: Proceedings of Annual Irish Conference on Artificial Intelligence and Cognitive Science ’92*, p. 342–345, Limerick, Ireland. Springer-Verlag.
1256. Quinlan J. R. (1979) Discovering rules from large collections of examples: A case study. In Michie D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.
1257. Quinlan J. R. (1986) Induction of decision trees. *Machine Learning*, 1, 81–106.
1258. Quinlan J. R. (1990) Learning logical definitions from relations. *Machine Learning*, 5(3), p. 239–266.
1259. Quinlan J. R. (1993) *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, California.
1260. Quinlan J. R. and Cameron-Jones R. M. (1993) FOIL: a midterm report. In Brazdil P. B. (Ed.), *European Conference on Machine Learning Proceedings (ECML-93)*, p. 3–20, Vienna. Springer-Verlag.
1261. Quirk R., Greenbaum S., Leech G., and Svartvik J. (1985) *A Comprehensive Grammar of the English Language*. Longman, New York.
1262. Rabani Y., Rabinovich Y., and Sinclair A. (1998) A computational view of population genetics. *Random Structures and Algorithms*, 12(4), p. 313–334.
1263. Rabiner L. R. and Juang B.-H. (1993) *Fundamentals of Speech Recognition*. Prentice-Hall, Upper Saddle River, New Jersey.
1264. Ramakrishnan R. and Ullman J. D. (1995) A survey of research in deductive database systems. *Journal of Logic Programming*, 23(2), p. 125–149.
1265. Ramsey F. P. (1931) Truth and probability. In Braithwaite R. B. (Ed.), *The Foundations of Mathematics and Other Logical Essays*. Harcourt Brace Jovanovich, New York.
1266. Raphson J. (1690) *Analysis aequationum universalis*. Apud Abelem Swalle, London.
1267. Rassenti S., Smith V., and Bulfin R. (1982) A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, 13, p. 402–417.

1268. Ratner D. and Warmuth M. (1986) Finding a shortest solution for the  $n \times n$  extension of the 15-puzzle is intractable. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, p. 168–172, Philadelphia. Morgan Kaufmann.
1269. Rauch H. E., Tung F., and Striebel C. T. (1965) Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8), p. 1445–1450.
1270. Rechenberg I. (1965) Cybernetic solution path of an experimental problem. Library translation 1122, Royal Aircraft Establishment.
1271. Rechenberg I. (1973) *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany.
1272. Regin J. (1994) A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 362–367, Seattle. AAAI Press.
1273. Reichenbach H. (1949) *The Theory of Probability: An Inquiry into the Logical and Mathematical Foundations of the Calculus of Probability* (Second edition). University of California Press, Berkeley and Los Angeles.
1274. Reif J. (1979) Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, p. 421–427, San Juan, Puerto Rico. IEEE, IEEE Computer Society Press.
1275. Reiter E. and Dale R. (2000) *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK.
1276. Reiter R. (1980) A logic for default reasoning. *Artificial Intelligence*, 13(1–2), p. 81–132.
1277. Reiter R. (1991) The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz V. (Ed.) *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, p. 359–380. Academic Press, New York.
1278. Reiter R. (2001a) On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, 2(4), p. 433–457.
1279. Reiter R. (2001b) *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, Massachusetts.
1280. Reitman W. and Wilcox B. (1979) The structure and performance of the INTERIM.2 Go program. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, p. 711–719, Tokyo. IJCAII.
1281. Remus H. (1962) Simulation of a learning machine for playing Go. In *Proceedings IFIP Congress*, p. 428–432, Amsterdam, London, New York. Elsevier/North-Holland.
1282. Rényi A. (1970) *Probability Theory*. Elsevier/North-Holland, Amsterdam, London, New York.
1283. Rescher N. and Urquhart A. (1971) *Temporal Logic*. Springer-Verlag, Berlin.
1284. Reynolds C. W. (1987) Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21, p. 25–34. SIGGRAPH '87 Conference Proceedings.
1285. Rich E. and Knight K. (1991) *Artificial Intelligence* (second edition). McGraw-Hill, New York.
1286. Richardson M., Bilmes J., and Diorio C. (2000) Hidden-articulator Markov models: Performance improvements and robustness to noise. In *ICASSP-2000: 2000 International Conference on Acoustics, Speech, and Signal Processing*, Los Alamitos, CA. IEEE Computer Society Press.
1287. Rieger C. (1976) An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence*, 7, p. 89–127.
1288. Ringle M. (1979) *Philosophical Perspectives in Artificial Intelligence*. Humanities Press, Atlantic Highlands, New Jersey.
1289. Rintanen J. (1999) Improvements to the evaluation of quantified boolean formulae. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, p. 1192–1197, Stockholm. Morgan Kaufmann.

1290. Ripley B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK.
1291. Rissanen J. (1984) Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, *IT-30*(4), p. 629–636.
1292. Ritchie G. D. and Hanna F. K. (1984) AM: A case study in AI methodology. *Artificial Intelligence*, *23*(3), p. 249–268.
1293. Rivest R. (1987) Learning decision lists. *Machine Learning*, *2*(3), p. 229–246.
1294. Roberts L. G. (1963) Machine perception of three-dimensional solids. Technical report 315, MIT Lincoln Laboratory.
1295. Robertson N. and Seymour P. D. (1986) Graph minors. ii. Algorithmic aspects of tree-width. *Journal of Algorithms*, *7*(3), p. 309–322.
1296. Robertson S. E. (1977) The probability ranking principle in ir. *Journal of Documentation*, *33*, p. 294–304.
1297. Robertson S. E. and Sparck Jones K. (1976) Relevance weighting of search terms. *Journal of the American Society for Information Science*, *27*, p. 129–146.
1298. Robinson J. A. (1965) A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, *12*, p. 23–41.
1299. Roche E. and Schabes Y. (1997) *Finite-State Language Processing (Language, Speech and Communication)*. Bradford Books, Cambridge.
1300. Rock I. (1984) *Perception*. W. H. Freeman, New York.
1301. Rorty R. (1965) Mind-body identity, privacy, and categories. *Review of Metaphysics*, *19*(1), p. 24–54.
1302. Rosenblatt F. (1957) The perceptron: A perceiving and recognizing automaton. Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Ithaca, New York.
1303. Rosenblatt F. (1960) On the convergence of reinforcement procedures in simple perceptrons. Report VG-1196-G-4, Cornell Aeronautical Laboratory, Ithaca, New York.
1304. Rosenblatt F. (1962) *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, Chicago.
1305. Rosenblatt M. (1956) Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, *27*, p. 832–837.
1306. Rosenblueth A., Wiener N., and Bigelow J. (1943) Behavior, purpose, and teleology. *Philosophy of Science*, *10*, p. 18–24.
1307. Rosenschein J. S. and Zlotkin G. (1994) *Rules of Encounter*. MIT Press, Cambridge, Massachusetts.
1308. Rosenschein S. J. (1985) Formal theories of knowledge in AI and robotics. *New Generation Computing*, *3*(4), p. 345–357.
1309. Rosenthal D. M. (Ed.) (1971) *Materialism and the Mind-Body Problem*. Prentice-Hall, Upper Saddle River, New Jersey.
1310. Ross S. M. (1988) *A First Course in Probability* (third edition). Macmillan, London.
1311. Roussel P. (1975) Prolog: Manual de reference et d'utilisation. Tech. rep., Groupe d'Intelligence Artificielle, Université d'Aix-Marseille.
1312. Rouveiro C. and Puget J.-F. (1989) A simple and general solution for inverting resolution. In *Proceedings of the European Working Session on Learning*, p. 201–210, Porto, Portugal. Pitman.
1313. Rowat P. F. (1979) *Representing the Spatial Experience and Solving Spatial problems in a Simulated Robot Environment*. Ph.D. thesis, University of British Columbia, Vancouver, BC, Canada.
1314. Roweis S. T. and Ghahramani Z. (1999) A unifying review of Linear Gaussian Models. *Neural Computation*, *11*(2), p. 305–345.

1315. Rubin D. (1988) Using the SIR algorithm to simulate posterior distributions. In Bernardo J. M., de Groot M. H., Lindley D. V., and Smith A. F. M. (Eds.) *Bayesian Statistics 3*, p. 395–402. Oxford University Press, Oxford, UK.
1316. Rumelhart D. E., Hinton G. E., and Williams R. J. (1986a) Learning internal representations by error propagation. In Rumelhart D. E. and McClelland J. L. (Eds.) *Parallel Distributed Processing*, Vol. 1, chap. 8, p. 318–362. MIT Press, Cambridge, Massachusetts.
1317. Rumelhart D. E., Hinton G. E., and Williams R. J. (1986b) Learning representations by back-propagating errors. *Nature*, 323, p. 533–536.
1318. Rumelhart D. E. and McClelland J. L. (Eds.) (1986) *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts.
1319. Ruspini E. H., Lowrance J. D., and Strat T. M. (1992) Understanding evidential reasoning. *International Journal of Approximate Reasoning*, 6(3), p. 401–424.
1320. Russell J. G. B. (1990) Is screening for abdominal aortic aneurysm worthwhile? *Clinical Radiology*, 41, p. 182–184.
1321. Russell S. J. (1985) The compleat guide to MRS. Report STAN-CS-85-1080, Computer Science Department, Stanford University.
1322. Russell S. J. (1986) A quantitative analysis of analogy by similarity. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, p. 284–288, Philadelphia. Morgan Kaufmann.
1323. Russell S. J. (1988) Tree-structured bias. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Vol. 2, p. 641–645, St. Paul, Minnesota. Morgan Kaufmann.
1324. Russell S. J. (1992) Efficient memory-bounded search methods. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, p. 1–5, Vienna. Wiley.
1325. Russell S. J. (1998) Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual ACM Workshop on Computational Learning Theory (COLT-98)*, p. 101–103, Madison, Wisconsin. ACM Press.
1326. Russell S. J., Binder J., Koller D., and Kanazawa K. (1995) Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, p. 1146–52, Montreal. Morgan Kaufmann.
1327. Russell S. J. and Grosof B. (1987) A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle. Morgan Kaufmann.
1328. Russell S. J. and Norvig P. (1995) *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey.
1329. Russell S. J. and Subramanian D. (1995) Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 3, p. 575–609.
1330. Russell S. J., Subramanian D., and Parr R. (1993) Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, p. 338–345, Chambéry, France. Morgan Kaufmann.
1331. Russell S. J. and Wefald E. H. (1989) On optimal game-tree search using rational meta-reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, p. 334–340, Detroit. Morgan Kaufmann.
1332. Russell S. J. and Wefald E. H. (1991) *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, Massachusetts.
1333. Rustagi J. S. (1976) *Variational Methods in Statistics*. Academic Press, New York.
1334. Ryder J. L. (1971) Heuristic analysis of large trees as generated in the game of Go. Memo AIM-155, Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Stanford, California.
1335. Sabin D. and Freuder E. C. (1994) Contradicting conventional wisdom in constraint satisfaction. In *ECAI 94: 11th European Conference on Artificial Intelligence. Proceedings*, p. 125–129, Amsterdam. Wiley.

1336. Sacerdoti E. D. (1974) Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2), p. 115–135.
1337. Sacerdoti E. D. (1975) The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, p. 206–214, Tbilisi, Georgia. IJCAII.
1338. Sacerdoti E. D. (1977) *A Structure for Plans and Behavior*. Elsevier/North-Holland, Amsterdam, London, New York.
1339. Sacerdoti E. D., Fikes R. E., Reboh R., Sagalowicz D., Waldinger R., and Wilber B. M. (1976) QLISP — a language for the interactive development of complex systems. In *Proceedings of the AFIPS National Computer Conference*, p. 349–356.
1340. Sacks E. and Joskowicz L. (1993) Automated modeling and kinematic simulation of mechanisms. *Computer Aided Design*, 25(2), p. 106–118.
1341. Sadri F. and Kowalski R. (1995) Variants of the event calculus. In *International Conference on Logic Programming*, p. 67–81.
1342. Sag I. and Wasow T. (1999) *Syntactic Theory: An Introduction*. CSLI Publications, Stanford, California.
1343. Sager N. (1981) *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Massachusetts.
1344. Sahami M., Dumais S. T., Heckerman D., and Horvitz E. J. (1998) A Bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin. AAAI Technical Report WS-98-05.
1345. Sahami M., Hearst M. A., and Saund E. (1996) Applying the multiple cause mixture model to text categorization. In Saitta L. (Ed.), *Proceedings of ICML-96, 13th International Conference on Machine Learning*, p. 435–443, Bari, Italy. Morgan Kaufmann Publishers.
1346. Salomaa A. (1969) Probabilistic and weighted grammars. *Information and Control*, 15, p. 529–544.
1347. Salton G. and McGill M. J. (1983) *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY.
1348. Salton G., Wong A., and Yang C. S. (1975) A vector space model for automatic indexing. *Communications of the ACM*, 18(11), p. 613–620.
1349. Samuel A. L. (1959) Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3), p. 210–229.
1350. Samuel A. L. (1967) Some studies in machine learning using the game of checkers II — Recent progress. *IBM Journal of Research and Development*, 11(6), p. 601–617.
1351. Samuelsson C. and Rayner M. (1991) Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, p. 609–615, Sydney. Morgan Kaufmann.
1352. Sato T. and Kameya Y. (1997) PRISM: A symbolic-statistical modeling language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, p. 1330–1335, Nagoya, Japan. Morgan Kaufmann.
1353. Saul L. K., Jaakkola T., and Jordan M. I. (1996) Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4, p. 61–76.
1354. Savage L. J. (1954) *The Foundations of Statistics*. Wiley, New York.
1355. Sayre K. (1993) Three more flaws in the computational model. Paper presented at the APA (Central Division) Annual Conference, Chicago, Illinois.
1356. Schabes Y., Abeille A., and Joshi A. K. (1988) Parsing strategies with lexicalized grammars: Application to tree adjoining grammars. In Varga D. (Ed.), *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Vol. 2, p. 578–583, Budapest. John von Neumann Society for Computer Science.

1357. Schaeffer J. (1997) *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, Berlin.
1358. Schank R. C. and Abelson R. P. (1977) *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Associates, Potomac, Maryland.
1359. Schank R. C. and Riesbeck C. (1981) *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum Associates, Potomac, Maryland.
1360. Schapire R. E. (1999) Theoretical views of boosting and applications. In *Algorithmic Learning Theory: Proceedings of the 10th International Conference (ALT'99)*, p. 13–25. Springer-Verlag, Berlin.
1361. Schapire R. E. (1990) The strength of weak learnability. *Machine Learning*, 5(2), p. 197–227.
1362. Schmolze J. G. and Lipkis T. A. (1983) Classification in the KL-ONE representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, p. 330–332, Karlsruhe, Germany. Morgan Kaufmann.
1363. Schofield P. D. A. (1967) Complete solution of the eight puzzle. In Dale E. and Michie D. (Eds.), *Machine Intelligence 2*, p. 125–133. Elsevier/North-Holland, Amsterdam, London, New York.
1364. Schölkopf B. and Smola A. J. (2002) *Learning with Kernels*. MIT Press, Cambridge, Massachusetts.
1365. Schöning T. (1999) A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science*, p. 410–414, New York. IEEE Computer Society Press.
1366. Schoppers M. J. (1987) Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, p. 1039–1046, Milan. Morgan Kaufmann.
1367. Schoppers M. J. (1989) In defense of reaction plans as caches. *AI Magazine*, 10(4), p. 51–60.
1368. Schröder E. (1877) *Der Operationskreis des Logikkalküls*. B. G. Teubner, Leipzig.
1369. Schultz W., Dayan P., and Montague P. R. (1997) A neural substrate of prediction and reward. *Science*, 275, p. 1593.
1370. Schütze H. (1995) *Ambiguity in Language Learning: Computational and Cognitive Models*. Ph.D. thesis, Stanford University. Also published by CSLI Press, 1997.
1371. Schwartz J. T., Scharir M., and Hopcroft J. (1987) *Planning, Geometry and Complexity of Robot Motion*. Ablex Publishing Corporation, Norwood, NJ.
1372. Schwartz S. P. (Ed.) (1977) *Naming, Necessity, and Natural Kinds*. Cornell University Press, Ithaca, New York.
1373. Scott D. and Krauss P. (1966) Assigning probabilities to logical formulas. In Hintikka J. and Suppes P. (Eds.) *Aspects of Inductive Logic*. North-Holland, Amsterdam.
1374. Scriven M. (1953) The mechanical concept of mind. *Mind*, 62, p. 230–240.
1375. Searle J. R. (1969) *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, UK.
1376. Searle J. R. (1980) Minds, brains, and programs. *Behavioral and Brain Sciences*, 3, p. 417–457.
1377. Searle J. R. (1984) *Minds, Brains and Science*. Harvard University Press, Cambridge, Massachusetts.
1378. Searle J. R. (1990) Is the brain's mind a computer program? *Scientific American*, 262, p. 26–31.
1379. Searle J. R. (1992) *The Rediscovery of the Mind*. MIT Press, Cambridge, Massachusetts.
1380. Selman B., Kautz H., and Cohen B. (1996) Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26*, p. 521–532. American Mathematical Society, Providence, Rhode Island.
1381. Selman B. and Levesque H. J. (1993) The complexity of path-based defeasible inheritance. *Artificial Intelligence*, 62(2), p. 303–339.

1382. Selman B., Levesque H. J., and Mitchell D. (1992) A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, p. 440–446, San Jose. AAAI Press.
1383. Shachter R. D. (1986) Evaluating influence diagrams. *Operations Research*, 34, p. 871–882.
1384. Shachter R. D. (1998) Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, p. 480–487, Madison, Wisconsin. Morgan Kaufmann.
1385. Shachter R. D., D'Ambrosio B., and Del Favero B. A. (1990) Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, p. 126–131, Boston. MIT Press.
1386. Shachter R. D. and Kenley C. R. (1989) Gaussian influence diagrams. *Management Science*, 35(5), p. 527–550.
1387. Shachter R. D. and Peot M. (1989) Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario. Morgan Kaufmann.
1388. Shafer G. (1976) *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey.
1389. Shafer G. and Pearl J. (Eds.) (1990) *Readings in Uncertain Reasoning*. Morgan Kaufmann, San Mateo, California.
1390. Shahookar K. and Mazumder P. (1991) VLSI cell placement techniques. *Computing Surveys*, 23(2), p. 143–220.
1391. Shanahan M. (1997) *Solving the Frame Problem*. MIT Press, Cambridge, Massachusetts.
1392. Shanahan M. (1999) The event calculus explained. In Wooldridge M. J. and Veloso M. (Eds.) *Artificial Intelligence Today*, p. 409–430. Springer-Verlag, Berlin.
1393. Shankar N. (1986) *Proof-Checking Metamathematics*. Ph.D. thesis, Computer Science Department, University of Texas at Austin.
1394. Shannon C. E. and Weaver W. (1949) *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois.
1395. Shannon C. E. (1950) Programming a computer for playing chess. *Philosophical Magazine*, 41(4), p. 256–275.
1396. Shapiro E. (1981) An algorithm that infers theories from facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, p. 1064, Vancouver, British Columbia. Morgan Kaufmann.
1397. Shapiro S. C. (Ed.) (1992) *Encyclopedia of Artificial Intelligence* (second edition). Wiley, New York.
1398. Shapley S. (1953) Stochastic games. In *Proceedings of the National Academy of Sciences*, Vol. 39, p. 1095–1100.
1399. Shavlik J. and Dietterich T. (Eds.) (1990) *Readings in Machine Learning*. Morgan Kaufmann, San Mateo, California.
1400. Shelley M. (1818) *Frankenstein: or, the Modern Prometheus*. Pickering and Chatto.
1401. Shenoy P. P. (1989) A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 3(5), p. 383–411.
1402. Shi J. and Malik J. (2000) Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8), p. 888–905.
1403. Shoham Y. (1987) Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33(1), p. 89–104.
1404. Shoham Y. (1993) Agent-oriented programming. *Artificial Intelligence*, 60(1), p. 51–92.
1405. Shoham Y. (1994) *Artificial Intelligence Techniques in Prolog*. Morgan Kaufmann, San Mateo, California.

1406. Shortliffe E. H. (1976) *Computer-Based Medical Consultations: MYCIN*. Elsevier/North-Holland, Amsterdam, London, New York.
1407. Shwe M. and Cooper G. (1991) An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. *Computers and Biomedical Research*, 1991(5), p. 453–475.
1408. Siekmann J. and Wrightson G. (Eds.) (1983) *Automation of Reasoning*. Springer-Verlag, Berlin.
1409. Sietsma J. and Dow R. J. F. (1988) Neural net pruning — why and how. In *IEEE International Conference on Neural Networks*, p. 325–333, San Diego. IEEE.
1410. Siklossy L. and Dreussi J. (1973) An efficient robot planner which generates its own procedures. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, p. 423–430, Stanford, California. IJCAI.
1411. Silverstein C., Henzinger M., Marais H., and Moricz M. (1998) Analysis of a very large altavista query log. Tech. rep. 1998-014, Digital Systems Research Center.
1412. Simmons R. and Koenig S. (1995) Probabilistic robot navigation in partially observable environments. In *Proceedings of IJCAI-95*, p. 1080–1087, Montreal, Canada. IJCAI, Inc.
1413. Simmons R. and Slocum J. (1972) Generating english discourse from semantic networks. *Communications of the ACM*, 15(10), p. 891–905.
1414. Simon H. A. (1947) *Administrative behavior*. Macmillan, New York.
1415. Simon H. A. (1957) *Models of Man: Social and Rational*. John Wiley, New York.
1416. Simon H. A. (1963) Experiments with a heuristic compiler. *Journal of the Association for Computing Machinery*, 10, p. 493–506.
1417. Simon H. A. (1981) *The Sciences of the Artificial* (second edition). MIT Press, Cambridge, Massachusetts.
1418. Simon H. A. (1982) *Models of Bounded Rationality, Volume 1*. The MIT Press, Cambridge, Massachusetts.
1419. Simon H. A. and Newell A. (1958) Heuristic problem solving: The next advance in operations research. *Operations Research*, 6, p. 1–10.
1420. Simon H. A. and Newell A. (1961) Computer simulation of human thinking and problem solving. *Datamation, June/July*, p. 35–37.
1421. Simon J. C. and Dubois O. (1989) Number of solutions to satisfiability instances — applications to knowledge bases. *Int. J. Pattern Recognition and Artificial Intelligence*, 3, p. 53–65.
1422. Sirovitch L. and Kirby M. (1987) Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 2, p. 586–591.
1423. Skinner B. F. (1953) *Science and Human Behavior*. Macmillan, London.
1424. Skolem T. (1920) Logisch-kombinatorische Untersuchungen Über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über die dichte Mengen. *Videnskapsselskapets skrifter I. Matematisk-naturvidenskabelig klasse*, 4.
1425. Skolem T. (1928) Über die mathematische Logik. *Norsk matematisk tidsskrift*, 10, p. 125–142.
1426. Slagle J. R. (1963a) A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the Association for Computing Machinery*, 10(4).
1427. Slagle J. R. (1963b) Game trees,  $m$  &  $n$  minimaxing, and the  $m$  &  $n$  alpha-beta procedure. Artificial intelligence group report 3, University of California, Lawrence Radiation Laboratory, Livermore, California.
1428. Slagle J. R. and Dixon J. K. (1969) Experiments with some programs that search game trees. *Journal of the Association for Computing Machinery*, 16(2), 189–207.
1429. Slate D. J. and Atkin L. R. (1977) CHESS 4.5 — Northwestern University chess program. In Frey P. W. (Ed.) *Chess Skill in Man and Machine*, p. 82–118. Springer-Verlag, Berlin.

1430. Slater E. (1950) Statistics for the chess computer and the factor of mobility. In *Symposium on Information Theory*, p. 150–152, London. Ministry of Supply.
1431. Sleator D. and Temperley D. (1993) Parsing English with a link grammar. In *Third Annual Workshop on Parsing technologies*.
1432. Sloman A. (1978) *The Computer Revolution in Philosophy*. Harvester Press, Hassocks, Sussex, UK.
1433. Smallwood R. D. and Sondik E. J. (1973) The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21, p. 1071–1088.
1434. Smith D. E., Genesereth M. R., and Ginsberg M. L. (1986) Controlling recursive inference. *Artificial Intelligence*, 30(3), p. 343–389.
1435. Smith D. R. (1990) KIDS: a semiautomatic program development system. *IEEE Transactions on Software Engineering*, 16(9), p. 1024–1043.
1436. Smith D. R. (1996) Machine support for software development. In *Proceedings of the 18th International Conference on Software Engineering*, p. 167–168, Berlin. IEEE Computer Society Press.
1437. Smith D. E. and Weld D. S. (1998) Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 889–896, Madison, Wisconsin. AAAI Press.
1438. Smith J. Q. (1988) *Decision Analysis*. Chapman and Hall, London.
1439. Smith J. M. and Szathmáry E. (1999) *The Origins of Life: From the Birth of Life to the Origin of Language*. Oxford University Press, Oxford, UK.
1440. Smith R. C. and Cheeseman P. (1986) On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4), p. 56–68.
1441. Smith S. J. J., Nau D. S., and Throop T. A. (1998) Success in spades: Using ai planning techniques to win the world championship of computer bridge. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 1079–1086, Madison, Wisconsin. AAAI Press.
1442. Smolensky P. (1988) On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 2, p. 1–74.
1443. Smyth P., Heckerman D., and Jordan M. I. (1997) Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9(2), p. 227–269.
1444. Soderland S. and Weld D. S. (1991) Evaluating nonlinear planning. Technical report TR-91-02-03, University of Washington Department of Computer Science and Engineering, Seattle, Washington.
1445. Solomonoff R. J. (1964) A formal theory of inductive inference. *Information and Control*, 7, p. 1–22; 224–254.
1446. Sondik E. J. (1971) *The Optimal Control of Partially Observable Markov Decision Processes*. Ph.D. thesis, Stanford University, Stanford, California.
1447. Sosic R. and Gu J. (1994) Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, 6(5), p. 661–668.
1448. Sowa J. (1999) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Blackwell, Oxford, UK.
1449. Spiegelhalter D. J. (1986) Probabilistic reasoning in predictive expert systems. In Kanal L. N. and Lemmer J. F. (Eds.) *Uncertainty in Artificial Intelligence*, p. 47–67. Elsevier/North-Holland, Amsterdam, London, New York.
1450. Spiegelhalter D. J., Dawid P., Lauritzen S., and Cowell R. (1993) Bayesian analysis in expert systems. *Statistical Science*, 8, p. 219–282.
1451. Spielberg S. (2001) AI. Кинофильм.
1452. Spirtes P., Glymour C., and Scheines R. (1993) *Causation, prediction, and search*. Springer-Verlag, Berlin.
1453. Springsteen B. (1992) 57 channels (and nothin' on). In *Human Touch*. Sony.

1454. Srinivasan A., Muggleton S. H., King R. D., and Sternberg M. J. E. (1994) Mutagenesis: ILP experiments in a non-determinate biological domain. In Wrobel S. (Ed.) *Proceedings of the 4th International Workshop on Inductive Logic Programming*, Vol. 237, p. 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.
1455. Srivastava M. and Bickford M. (1990) Formal verification of a pipelined microprocessor. *IEEE Software*, 7(5), p. 52–64.
1456. Stallman R. M. and Sussman G. J. (1977) Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2), p. 135–196.
1457. Stanfill C. and Waltz D. (1986) Toward memory-based reasoning. *Communications of the Association for Computing Machinery*, 29(12), p. 1213–1228.
1458. Stefik M. (1995) *Introduction to Knowledge Systems*. Morgan Kaufmann, San Mateo, California.
1459. Stein L. A. (2002) *Interactive Programming in Java (pre-publication draft)*. Morgan Kaufmann, San Mateo, California.
1460. Steinbach M., Karypis G., and Kumar V. (2000) A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, p. 109–110. ACM Press.
1461. Stevens K. A. (1981) The information content of texture gradients. *Biological Cybernetics*, 42, p. 95–105.
1462. Stickel M. E. (1985) Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4), p. 333–355.
1463. Stickel M. E. (1988) A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4, p. 353–380.
1464. Stiller L. B. (1992) KQNKRR. *JCCA Journal*, 15(1), p. 16–18.
1465. Stillings N. A., Weisler S., Feinstein M. H., Garfield J. L., and Rissland E. L. (1995) *Cognitive Science: An Introduction* (second edition). MIT Press, Cambridge, Massachusetts.
1466. Stockman G. (1979) A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12(2), p. 179–196.
1467. Stolcke A. and Omohundro S. (1994) Inducing probabilistic grammars by Bayesian model merging. In *Proceedings of the Second International Colloquium on Grammatical Inference and Applications (ICGI-94)*, p. 106–118, Alicante, Spain. Springer-Verlag.
1468. Stone P. (2000) *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*. MIT Press, Cambridge, Massachusetts.
1469. Strachey C. (1952) Logical or non-mathematical programmes. In *Proceedings of the Association for Computing Machinery (ACM)*, p. 46–49, Toronto, Canada.
1470. Subramanian D. (1993) Artificial intelligence and conceptual design. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, p. 800–809, Chambéry, France. Morgan Kaufmann.
1471. Subramanian D. and Feldman R. (1990) The utility of EBL in recursive domain theories. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, p. 942–949, Boston. MIT Press.
1472. Subramanian D. and Wang E. (1994) Constraint-based kinematic synthesis. In *Proceedings of the International Conference on Qualitative Reasoning*, p. 228–239. AAAI Press.
1473. Sugihara K. (1984) A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(5), p. 578–586.
1474. Sussman G. J. (1975) *A Computer Model of Skill Acquisition*. Elsevier/North-Holland, Amsterdam, London, New York.
1475. Sussman G. J. and Winograd T. (1970) MICRO-PLANNER Reference Manual. Ai memo 203, MIT AI Lab, Cambridge, Massachusetts.
1476. Sutherland I. (1963) Sketchpad: A man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference*, p. 329–346. IFIPS.

1477. Sutton R. S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning*, 3, p. 9–44.
1478. Sutton R. S., McAllester D. A., Singh S. P., and Mansour Y. (2000) Policy gradient methods for reinforcement learning with function approximation. In Solla S. A., Leen T. K., and Müller K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*, p. 1057–1063. MIT Press, Cambridge, Massachusetts.
1479. Sutton R. S. (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning: Proceedings of the Seventh International Conference*, p. 216–224, Austin, Texas. Morgan Kaufmann.
1480. Sutton R. S. and Barto A. G. (1998) *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts.
1481. Swade D. D. (1993) Redeeming Charles Babbage's mechanical computer. *Scientific American*, 268(2), p. 86–91.
1482. Swerling P. (1959) First order error propagation in a stagewise smoothing procedure for satellite observations. *Journal of Astronautical Sciences*, 6, p. 46–52.
1483. Swift T. and Warren D. S. (1994) Analysis of SLG-WAM evaluation of definite programs. In *Logic Programming. Proceedings of the 1994 International Symposium*, p. 219–235, Ithaca, NY. MIT Press.
1484. Syrjänen T. (2000) Lparse 1.0 user's manual.  
<http://saturn.tcs.hut.fi/Software/smodels>.
1485. Tadepalli P. (1993) Learning from queries and examples with tree-structured bias. In *Proceedings of the Tenth International Conference on Machine Learning*, p. 322–329, Amherst, Massachusetts. Morgan Kaufmann.
1486. Tait P. G. (1880) Note on the theory of the “15 puzzle”. *Proceedings of the Royal Society of Edinburgh*, 10, p. 664–665.
1487. Tamaki H. and Sato T. (1986) OLD resolution with tabulation. In *Third International Conference on Logic Programming*, p. 84–98, London. Springer-Verlag.
1488. Tambe M., Newell A., and Rosenbloom P. S. (1990) The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5, p. 299–348.
1489. Tarjan R. E. (1983) *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM (Society for Industrial and Applied Mathematics, Philadelphia).
1490. Tarski A. (1935) Die Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1, p. 261–405.
1491. Tarski A. (1956) *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Oxford University Press, Oxford, UK.
1492. Tash J. K. and Russell S. J. (1994) Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1079–1085, Seattle. AAAI Press.
1493. Tate A. (1975a) Interacting goals and their use. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, p. 215–218, Tbilisi, Georgia. IJCAII.
1494. Tate A. (1975b) *Using Goal Structure to Direct Search in a Problem Solver*. Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.
1495. Tate A. (1977) Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, p. 888–893, Cambridge, Massachusetts. IJCAII.
1496. Tate A. and Whiter A. M. (1984) Planning with multiple resource constraints and an application to a naval planning problem. In *Proceedings of the First Conference on AI Applications*, p. 410–416, Denver, Colorado.
1497. Tatman J. A. and Shachter R. D. (1990) Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2), p. 365–379.
1498. Tesauro G. (1989) Neurogammon wins computer olympiad. *Neural Computation*, 1(3), p. 321–323.

1499. Tesauro G. (1992) Practical issues in temporal difference learning. *Machine Learning*, 8(3–4), p. 257–277.
1500. Tesauro G. (1995) Temporal difference learning and TD-Gammon. *Communications of the Association for Computing Machinery*, 38(3), p. 58–68.
1501. Tesauro G. and Sejnowski T. (1989) A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3), p. 357–390.
1502. Thagard P. (1996) *Mind: Introduction to Cognitive Science*. MIT Press, Cambridge, Massachusetts.
1503. Thaler R. (1992) *The Winner's Curse: Paradoxes and Anomalies of Economic Life*. Princeton University Press, Princeton, New Jersey.
1504. Thielscher M. (1999) From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2), p. 277–299.
1505. Thomason R. H. (Ed.) (1974) *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, Connecticut.
1506. Thompson D. W. (1917) *On Growth and Form*. Cambridge University Press, Cambridge, UK.
1507. Thrun S. (2000) Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA. IEEE.
1508. Thrun S. (2002) Robotic mapping: A survey. In Lakemeyer G. and Nebel B. (Eds.), *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann. Готовится к выпуску.
1509. Titterington D. M., Smith A. F. M., and Makov U. E. (1985) *Statistical analysis of finite mixture distributions*. Wiley, New York.
1510. Toffler A. (1970) *Future Shock*. Bantam.
1511. Tomasi C. and Kanade T. (1992) Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 9, p. 137–154.
1512. Touretzky D. S. (1986) *The Mathematics of Inheritance Systems*. Pitman and Morgan Kaufmann, London and San Mateo, California.
1513. Trucco E. and Verri A. (1998) *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, Upper Saddle River, New Jersey.
1514. Tsang E. (1993) *Foundations of Constraint Satisfaction*. Academic Press, New York.
1515. Tsitsiklis J. N. and Van Roy B. (1997) An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5), p. 674–690.
1516. Turner K. and Wolpert D. (2000) Collective intelligence and braess' paradox. In *Proceedings of the AAAI/IAAI*, p. 104–109.
1517. Turcotte M., Muggleton S. H., and Sternberg M. J. E. (2001) Automated discovery of structural signatures of protein fold and function. *Journal of Molecular Biology*, 306, p. 591–605.
1518. Turing A. (1936) On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2nd series, 42, p. 230–265.
1519. Turing A. (1948) Intelligent machinery. Tech. rep., National Physical Laboratory. Перепечатано в Ince D. C. (Ed.) (1992) *The Collected Works of A. M. Turing. Volume 3*. North-Holland, Amsterdam.
1520. Turing A. (1950) Computing machinery and intelligence. *Mind*, 59, p. 433–460.
1521. Turing A., Strachey C., Bates M. A., and Bowden B. V. (1953) Digital computers applied to games. In Bowden B. V. (Ed.) *Faster than Thought*, p. 286–310. Pitman, London.
1522. Turtle H. R. and Croft W. B. (1992) A comparison of text retrieval models. *The Computer Journal*, 35(1), p. 279–289.

1523. Tversky A. and Kahneman D. (1982) Causal schemata in judgements under uncertainty. In Kahneman D., Slovic P., and Tversky A. (Eds.) *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge, UK.
1524. Ullman J. D. (1985) Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, 10(3), p. 289–321.
1525. Ullman J. D. (1989) *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, Maryland.
1526. Ullman S. (1979) *The Interpretation of Visual Motion*. MIT Press, Cambridge, Massachusetts.
1527. Vaessens R. J. M., Aarts E. H. I., and Lenstra J. K. (1996) Job shop scheduling by local search. *INFORMS J. on Computing*, 8, p. 302–117.
1528. Valiant L. (1984) A theory of the learnable. *Communications of the Association for Computing Machinery*, 27, p. 1134–1142.
1529. van Benthem J. (1983) *The Logic of Time*. D. Reidel, Dordrecht, Netherlands.
1530. Van Emden M. H. and Kowalski R. (1976) The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, 23(4), p. 733–742.
1531. van Harmelen F. and Bundy A. (1988) Explanation-based generalisation = partial evaluation. *Artificial Intelligence*, 36(3), p. 401–412.
1532. van Heijenoort J. (Ed.) (1967) *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts.
1533. Van Hentenryck P., Saraswat V., and Deville Y. (1998) Design, implementation, and evaluation of the constraint language cc(fd). *Journal of Logic Programming*, 37(1–3), p. 139–164.
1534. van Nunen J. A. E. E. (1976) A set of successive approximation methods for discounted Markovian decision problems. *Zeitschrift für Operations Research, Serie A*, 20(5), p. 203–208.
1535. van Roy B. (1998) *Learning and value function approximation in complex decision processes*. Ph.D. thesis, Laboratory for Information and Decision Systems, MIT, Cambridge, Massachusetts.
1536. Van Roy P. L. (1990) Can logic programming execute as fast as imperative programming? Report UCB/CSD 90/600, Computer Science Division, University of California, Berkeley, California.
1537. Vapnik V. N. (1998) *Statistical Learning Theory*. Wiley, New York.
1538. Vapnik V. N. and Chervonenkis A. Y. (1971) On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16, p. 264–280.
1539. Varian H. R. (1995) Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce*, p. 13–21.
1540. Veloso M. and Carbonell J. G. (1993) Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10, p. 249–278.
1541. Vere S. A. (1983) Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 5, p. 246–267.
1542. Vinge V. (1993) The coming technological singularity: How to survive in the post-human era. In *VISION-21 Symposium*. NASA Lewis Research Center and the Ohio Aerospace Institute.
1543. Viola P. and Jones M. (2002) Robust real-time object detection. *International Journal of Computer Vision*. В печати.
1544. von Mises R. (1928) *Wahrscheinlichkeit, Statistik und Wahrheit*. J. Springer, Berlin.
1545. von Neumann J. (1928) Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, p. 295–320.
1546. von Neumann J. and Morgenstern O. (1944) *Theory of Games and Economic Behavior* (first edition). Princeton University Press, Princeton, New Jersey.
1547. von Winterfeldt D. and Edwards W. (1986) *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge, UK.

1548. Voorhees E. M. (1993) Using WordNet to disambiguate word senses for text retrieval. In *Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 171–80, Pittsburgh. Association for Computing Machinery.
1549. Vossen T., Ball M., Lotem A., and Nau D. S. (2001) Applying integer programming to ai planning. *Knowledge Engineering Review*, 16, p. 85–100.
1550. Waibel A. and Lee K.-F. (1990) *Readings in Speech Recognition*. Morgan Kaufmann, San Mateo, California.
1551. Waldinger R. (1975) Achieving several goals simultaneously. In Elcock E. W. and Michie D. (Eds.) *Machine Intelligence 8*, p. 94–138. Ellis Horwood, Chichester, England.
1552. Waltz D. (1975) Understanding line drawings of scenes with shadows. In Winston P. H. (Ed.) *The Psychology of Computer Vision*. McGraw-Hill, New York.
1553. Wanner E. (1974) *On remembering, forgetting and understanding sentences*. Mouton, The Hague and Paris.
1554. Warren D. H. D. (1974) WARPLAN: A System for Generating Plans. Department of Computational Logic Memo 76, University of Edinburgh, Edinburgh, Scotland.
1555. Warren D. H. D. (1976) Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, p. 344–354.
1556. Warren D. H. D. (1983) An abstract Prolog instruction set. Technical note 309, SRI International, Menlo Park, California.
1557. Warren D. H. D., Pereira L. M., and Pereira F. (1977) PROLOG: The language and its implementation compared with LISP. *SIGPLAN Notices*, 12(8), p. 109–115.
1558. Watkins C. J. (1989) *Models of Delayed Reinforcement Learning*. Ph.D. thesis, Psychology Department, Cambridge University, Cambridge, UK.
1559. Watson J. D. and Crick F. H. C. (1953) A structure for deoxyribose nucleic acid. *Nature*, 171, p. 737.
1560. Webber B. L. (1983) So what can we talk about now? In Brady M. and Berwick R. (Eds.) *Computational Models of Discourse*. MIT Press, Cambridge, Massachusetts.
1561. Webber B. L. (1988) Tense as discourse anaphora. *Computational Linguistics*, 14(2), p. 61–73.
1562. Webber B. L. and Nilsson N. J. (Eds.) (1981) *Readings in Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.
1563. Weidenbach C. (2001) SPASS: Combining superposition, sorts and splitting. In Robinson A. and Voronkov A. (Eds.) *Handbook of Automated Reasoning. Volume II*. p. 1965–2013, Elsevier Science.
1564. Weiss G. (1999) *Multagent systems*. MIT Press, Cambridge, Massachusetts.
1565. Weiss S. and Kulikowski C. A. (1991) *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann, San Mateo, California.
1566. Weizenbaum J. (1976) *Computer Power and Human Reason*. W. H. Freeman, New York.
1567. Weld D. S. (1994) An introduction to least commitment planning. *AI Magazine*, 15(4), p. 27–61.
1568. Weld D. S. (1999) Recent advances in ai planning. *AI Magazine*, 20(2), p. 93–122.
1569. Weld D. S., Anderson C. R., and Smith D. E. (1998) Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 897–904, Madison, Wisconsin. AAAI Press.
1570. Weld D. S. and de Kleer J. (1990) *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, California.
1571. Weld D. S. and Etzioni O. (1994) The first law of robotics: A call to arms. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle. AAAI Press.
1572. Wellman M. P. (1985) Reasoning about preference models. Technical report MIT/LCS/TR-340, Laboratory for Computer Science, MIT, Cambridge, Massachusetts.

1573. Wellman M. P. (1988) *Formulation of Tradeoffs in Planning under Uncertainty*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
1574. Wellman M. P. (1990a) Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44(3), p. 257–303.
1575. Wellman M. P. (1990b) The STRIPS assumption for planning under uncertainty. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, p. 198–203, Boston. MIT Press.
1576. Wellman M. P. (1995) The economic approach to artificial intelligence. *ACM Computing Surveys*, 27(3), p. 360–362.
1577. Wellman M. P., Breese J. S., and Goldman R. (1992) From knowledge bases to decision models. *Knowledge Engineering Review*, 7(1), p. 35–53.
1578. Wellman M. P. and Doyle J. (1992) Modular utility representation for decision-theoretic planning. In *Proceedings, First International Conference on AI Planning Systems*, p. 236–242, College Park, Maryland. Morgan Kaufmann.
1579. Werbos P. (1974) *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University, Cambridge, Massachusetts.
1580. Werbos P. (1977) Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22, p. 25–38.
1581. Wesley M. A. and Lozano-Perez T. (1979) An algorithm for planning collision-free paths among polyhedral objects. *Communications of the ACM*, 22(10), p. 560–570.
1582. Wheatstone C. (1838) On some remarkable, and hitherto unresolved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London*, 2, p. 371–394.
1583. Whitehead A. N. (1911) *An Introduction to Mathematics*. Williams and Northgate, London.
1584. Whitehead A. N. and Russell B. (1910) *Principia Mathematica*. Cambridge University Press, Cambridge, UK.
1585. Whorf B. (1956) *Language, Thought, and Reality*. MIT Press, Cambridge, Massachusetts.
1586. Widrow B. (1962) Generalization and information storage in networks of adaline “neurons”. In Yovits M. C., Jacobi G. T., and Goldstein G. D. (Eds.) *Self-Organizing Systems 1962*, p. 435–461, Chicago, Illinois. Spartan.
1587. Widrow B. and Hoff M. E. (1960) Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, p. 96–104, New York.
1588. Wiener N. (1942) The extrapolation, interpolation, and smoothing of stationary time series. Osrd 370, Report to the Services 19, Research Project DIC-6037, MIT, Cambridge, Massachusetts.
1589. Wiener N. (1948) *Cybernetics*. Wiley, New York.
1590. Wilensky R. (1978) *Understanding goal-based stories*. Ph.D. thesis, Yale University, New Haven, Connecticut.
1591. Wilensky R. (1983) *Planning and Understanding*. Addison-Wesley, Reading, Massachusetts.
1592. Wilkins D. E. (1980) Using patterns and plans in chess. *Artificial Intelligence*, 14(2), p. 165–203.
1593. Wilkins D. E. (1988) *Practical Planning: Extending the AI Planning Paradigm*. Morgan Kaufmann, San Mateo, California.
1594. Wilkins D. E. (1990) Can AI planners solve practical problems? *Computational Intelligence*, 6(4), p. 232–246.
1595. Wilkins D. E., Myers K. L., Lowrance J. D., and Wesley L. P. (1995) Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1), p. 197–227.
1596. Wilks Y. (1975) An intelligent analyzer and understander of English. *Communications of the ACM*, 18(5), p. 264–274.

1597. Williams R. J. (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, p. 229–256.
1598. Williams R. J. and Baird L. C. I. (1993) Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep. NU-CCS-93-14, College of Computer Science, Northeastern University, Boston.
1599. Wilson R. A. and Keil F. C. (Eds.) (1999) *The MIT Encyclopedia of the Cognitive Sciences*. MIT Press, Cambridge, Massachusetts.
1600. Winograd S. and Cowan J. D. (1963) *Reliable Computation in the Presence of Noise*. MIT Press, Cambridge, Massachusetts.
1601. Winograd T. (1972) Understanding natural language. *Cognitive Psychology*, 3(1), p. 1–191.
1602. Winston P. H. (1970) Learning structural descriptions from examples. Technical report MAC-TR-76, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.
1603. Winston P. H. (1992) *Artificial Intelligence* (Third edition). Addison-Wesley, Reading, Massachusetts.
1604. Wirth R. and O'Rorke P. (1991) Constraints on predicate invention. In *Machine Learning: Proceedings of the Eighth International Workshop (ML-91)*, p. 457–461, Evanston, Illinois. Morgan Kaufmann.
1605. Witten I. H. and Bell T. C. (1991) The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4), p. 1085–1094.
1606. Witten I. H., Moffat A., and Bell T. C. (1999) *Managing Gigabytes: Compressing and Indexing Documents and Images* (second edition). Morgan Kaufmann, San Mateo, California.
1607. Wittgenstein L. (1922) *Tractatus Logico-Philosophicus* (second edition). Routledge and Kegan Paul, London. Перепечатано в 1971 году под редакцией Д.Ф. Пирса (D. F. Pears) и Б.Ф. Макгиннеса (B. F. McGuinness). На титульных листах этого английского перевода приведен оригинальный текст Витгенштейна на немецком языке; кроме того, включено предисловие Бертрана Рассела к изданию 1922 года.
1608. Wittgenstein L. (1953) *Philosophical Investigations*. Macmillan, London.
1609. Wojciechowski W. S. and Wojcik A. S. (1983) Automated design of multiple-valued logic circuits by automated theorem proving techniques. *IEEE Transactions on Computers*, C-32(9), p. 785–798.
1610. Wojcik A. S. (1983) Formal design verification of digital systems. In *ACM IEEE 20th Design Automation Conference Proceedings*, p. 228–234, Miami Beach, Florida. IEEE.
1611. Wood M. K. and Dantzig G. B. (1949) Programming of interdependent activities. i. general discussion. *Econometrica*, 17, p. 193–199.
1612. Woods W. A. (1973) Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings*, Vol. 42, p. 441–450.
1613. Woods W. A. (1975) What's in a link? Foundations for semantic networks. In Bobrow D. G. and Collins A. M. (Eds.) *Representation and Understanding: Studies in Cognitive Science*, p. 35–82. Academic Press, New York.
1614. Woods W. A. (1978) Semantics and quantification in natural language question answering. In *Advances in Computers*. Academic Press.
1615. Wooldridge M. and Rao A. (Eds.) (1999) *Foundations of rational agency*. Kluwer, Dordrecht, Netherlands.
1616. Wos L., Carson D., and Robinson G. (1964) The unit preference strategy in theorem proving. In *Proceedings of the Fall Joint Computer Conference*, p. 615–621.
1617. Wos L., Carson D., and Robinson G. (1965) Efficiency and completeness of the set-of-support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12, p. 536–541.
1618. Wos L., Overbeek R., Lusk E., and Boyle J. (1992) *Automated Reasoning: Introduction and Applications* (second edition). McGraw-Hill, New York.

1619. Wos L. and Robinson G. (1968) Paramodulation and set of support. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, p. 276–310. Springer-Verlag.
1620. Wos L., Robinson G., Carson D., and Shalla L. (1967) The concept of demodulation in theorem proving. *Journal of the Association for Computing Machinery*, 14, p. 698–704.
1621. Wos L. and Winker S. (1983) Open questions solved with the assistance of AURA. In Bledsoe W. W. and Loveland D. (Eds.) *Automated Theorem Proving: After 25 Years: Proceedings of the Special Session of the 89th Annual Meeting of the American Mathematical Society*, p. 71–88, Denver, Colorado. American Mathematical Society.
1622. Wright S. (1921) Correlation and causation. *Journal of Agricultural Research*, 20, p. 557–585.
1623. Wright S. (1931) Evolution in Mendelian populations. *Genetics*, 16, p. 97–159.
1624. Wright S. (1934) The method of path coefficients. *Annals of Mathematical Statistics*, 5, p. 161–215.
1625. Wu D. (1993) Estimating probability distributions over hypotheses with variable unification. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, p. 790–795, Chambéry, France. Morgan Kaufmann.
1626. Wygant R. M. (1989) CLIPS — a powerful development and delivery expert system tool. *Computers and Industrial Engineering*, 17, p. 546–549.
1627. Yamada K. and Knight K. (2001) A syntax-based statistical translation model. In *Proceedings of the Thirty Ninth Annual Conference of the Association for Computational Linguistics*, p. 228–235.
1628. Yang Q. (1990) Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6, p. 12–24.
1629. Yang Q. (1997) *Intelligent planning: A decomposition and abstraction based approach*. Springer-Verlag, Berlin.
1630. Yedidia J., Freeman W., and Weiss Y. (2001) Generalized belief propagation. In Leen T. K., Dietterich T., and Tresp V. (Eds.) *Advances in Neural Information Processing Systems 13*. MIT Press, Cambridge, Massachusetts.
1631. Yip K. M.-K. (1991) *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press, Cambridge, Massachusetts.
1632. Yngve V. (1955) A model and an hypothesis for language structure. In Locke W. N. and Booth A. D. (Eds.) *Machine Translation of Languages*, p. 208–226. MIT Press, Cambridge, Massachusetts.
1633. Yob G. (1975) Hunt the wumpus! *Creative Computing*, Sep/Oct.
1634. Yoshikawa T. (1990) *Foundations of Robotics: Analysis and Control*. MIT Press, Cambridge, Massachusetts.
1635. Young R. M., Pollack M. E., and Moore J. D. (1994) Decomposition and causality in partial order planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, p. 188–193, Chicago.
1636. Younger D. H. (1967) Recognition and parsing of context-free languages in time  $n^3$ . *Information and Control*, 10(2), p. 189–208.
1637. Zadeh L. A. (1965) Fuzzy sets. *Information and Control*, 8, p. 338–353.
1638. Zadeh L. A. (1978) Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1, p. 3–28.
1639. Zaritskii V. S., Svetnik V. B., and Shimelevich L. I. (1975) Monte-Carlo technique in problems of optimal information processing. *Automation and Remote Control*, 36, p. 2015–22.
1640. Zelle J. and Mooney R. J. (1996) Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, p. 1050–1055.
1641. Zermelo E. (1913) Über Eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proceedings of the Fifth International Congress of Mathematicians*, Vol. 2, p. 501–504.

1642. Zermelo E. (1976) An application of set theory to the theory of chess-playing. *Firbush News*, 6, p. 37–42. Перевод на английский работы [1641].
1643. Zhang N. L. and Poole D. (1994) A simple approach to bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, p. 171–178, Banff, Alberta. Morgan Kaufmann.
1644. Zhang N. L. and Poole D. (1996) Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5, p. 301–328.
1645. Zhou R. and Hansen E. (2002) Memory-bounded A\* graph search. In *Proceedings of the 15th International Flairs Conference*.
1646. Zhu D. J. and Latombe J.-C. (1991) New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7(1), p. 9–20.
1647. Zilberstein S. and Russell S. J. (1996) Optimal composition of real-time systems. *Artificial Intelligence*, 83, p. 181–213.
1648. Zimmermann H.-J. (Ed.) (1999) *Practical applications of fuzzy technologies*. Kluwer, Dordrecht, Netherlands.
1649. Zimmermann H.-J. (2001) *Fuzzy Set Theory — And Its Applications* (Fourth edition). Kluwer, Dordrecht, Netherlands.
1650. Zobrist A. L. (1970) *Feature Extraction and Representation for Pattern Recognition and the Game of Go*. Ph.D. thesis, University of Wisconsin.
1651. Zuse K. (1945) The Plankalkyl. Report 175, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Germany.
1652. Zweig G. and Russell S. J. (1998) Speech recognition with dynamic Bayesian networks. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. 173–180, Madison, Wisconsin. AAAI Press.

## ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

---

### 1

1-совместимость, 222

### 2

2-совместимость, 222

### 3

3SAT

ThreeSAT, 212

3-совместимость, 222

### A

AAAI

American Association for AI, 72

ABO

Asymptotic Bounded Optimality, 1285

ADL

Action Description Language, 517

ADP

Adaptive Dynamic Programming, 1015; 1038

AFSM

Augmented Finite State Machine, 1226

AISB

Artificial Intelligence and Simulation of Behaviour, 73

AUV

Autonomous Underwater Vehicle, 1189

### B

BDD

Binary Decision Diagram, 557

BNF

Backus–Naur Form, 1049; 1297

BO

Bounded Optimality, 1284

### C

CCD

Charge-Coupled Device, 1143

CDP

Composite Decision Process, 200

CFG

Context-Free Grammar, 1134

CMAC

Cerebellar Model Articulation Controller, 1040

CMU

Carnegie–Mellon University, 55; 264

### CNF

Conjunctive Normal Form, 308; 410

### CSP

Constraint Satisfaction Problem, 210; 408

### CWA

Closed-World Assumption, 484

### D

### DARPA

Defense Advanced Research Project Agency, 70

### DART

Dynamic Analysis and Replanning, 70

### DB

Data Base, 172

### DBN

Dynamic Bayesian Network, 746

### DCG

Definite Clause Grammar, 1067

### DDN

Dynamic Decision Network, 836

### DEC

Digital Equipment Corporation, 65; 398

### DL

Decision List, 892

### DT

Decision Tree, 892

### E

### EBL

Explanation-Based Learning, 915

### EI

Existential Instantiation, 382

### EKF

Extended Kalman Filter, 1202

### EM

expectation–maximization, 1109

### EMNLP

Empirical Methods in Natural Language Processing, 1138

### EMV

Expected Monetary Value, 785

### Entscheidungsproblem

, 44

### EPAM

Elementary Perceiver And Memorizer, 896

**F**

- FIFO** First-In-First-Out, 127  
**FMP** Fine-Motion Planning, 1217  
**FSA** Finite-State Automaton, 1123

**G**

- GA** Genetic Algorithm, 182  
**GLIE** Greedy in the Limit of Infinite Exploration, 1023  
**GOFAI** Good Old-Fashioned AI, 1253  
**GPS** General Problem Solver, 57; 555  
Global Positioning System, 84; 1191  
**GRL** Generic Robot Language, 1230  
**G-множество**, 909

**H**

- HMM** Hidden Markov Model, 67  
**Homo sapiens**, 34  
**HPP** Heuristic Programming Project, 63  
**HTN** Hierarchical Task Network, 518

**I**

- IE** Iterative Expansion, 201  
**IJCAI** International Joint Conference on AI, 72  
**IKBS** Intelligent Knowledge-Based Systems, 65  
**ILP** Inductive Logic Programming, 917; 1091  
**Internet**, 67

**K**

- KB** Knowledge Base, 284  
**KBIL** Knowledge-Based Inductive Learning, 917  
**k-совместимость**, 222

**L**

- LFG** Lexical-Functional Grammar, 1094  
**LIFO** Last-In-First-Out, 130

**LP**

- Linear Programming, 202  
**LRTA\*** Learning Real-Time A\*, 196  
**LT** Logic Theorist, 56

**M**

- MA\*** Memory-bounded A\*, 165  
**MAC** Maintaining Arc Consistency, 220  
**MAP** Maximum A Posteriori, 948  
**MCC** Microelectronics and Computer Technology Corporation, 65  
**MCL** Monte Carlo Localization, 1199  
**MDL** Minimum Description Length, 898  
**MDP** Markov Decision Process, 46; 817; 1011; 1216  
**MEMS** Micro-ElectroMechanical System, 1279  
**MGU** Most General Unifier, 387  
**MIT** Massachussets Institute of Techngologies, 55  
**ML** Maximum Likelihood, 950  
**MPI** Mutual Preferential Independence, 794  
**MRV** Minimum Remaining Values, 218  
**MST** Minimum Spanning Tree, 206

**N**

- Newell A., 37  
**NLP** Natural Language Processing, 36  
**NP** Nondeterministic Polynomial, 1290  
Noun Phrase, 1049  
**NP-полнота**, 44  
**NP-полный**, 118; 212; 303; 305; 335; 500; 682; 1168; 1183; 1290; 1291; 1296

**P**

- PAC** Probably Approximately Correct, 889  
**PAC-обучение**, 889; 898; 922  
**PCFG** Probabilistic Context-Free Grammar, 1106

**PDDL**

Planning Domain Definition Language, 518

**PDP**

Parallel Distributed Processing, 1002

**PD-контроллер**, 1222**PEAS**

Performance, Environment, Actuators, Sensors, 83

**PID-контроллер**, 1223**POMDP**

Partially Observable MDP, 832; 1216

**POP**

Partial-Order Planning, 528

**PP**

Prepositional Phrase, 1055

**PSPACE-полный**, 543; 555**PUMA**

Programmable Universal Machine for Assembly, 1237

**Р-контроллер**, 1222**Q****QALY**

Quality-Adjusted Life Year, 790

**Q-обучение**, 1011; 1026**Q-функция**, 1011**R****RAPS**

Reactive Action Plan System, 1230

**RBFS**

Recursive Best-First Search, 163

**RBL**

Relevance-Based Learning, 916

**RelClause**

Relative Clause, 1055

**rete-алгоритм**, 397**rete-сеть**, 398**ROC**

Receiver Operating Characteristic, 1115

**RST**

Rhetorical Structure Theory, 1096

**S****S**

Sentence, 1055

**SIGART**

Special Interest Group in Artificial Intelligence, 73

**Simon H. A.**, 37**SLAM**

Simultaneous Localization And Mapping, 1203

**SLD**

Straight Line Distance, 155

**SMA\***

Simplified MA\*, 165

**sos**

set of support, 422; 424

**SRI**, 375

Stanford Research Institute, 58

**SSD**

Sum of Squared Differences, 1157

**SVM**

Support Vector Machine, 898

**S-множество**, 909**T****TAG**

Tree-Adjoining Grammar, 1094

**TD**

Temporal Difference, 1017

**TREC**

Text REtrieval Conference, 1120

**TSP**

Traveling Salesperson Problem, 121; 146; 206

**U****UAV**

Unmanned Air Vehicle, 1189

**UI**

Universal Instantiation, 381

**ULV**

Unmanned Land Vehicle, 1189

**UNA**

Unique Names Assumption, 484

**V****VC**

Vapnik–Chervonenkis, 898

**VC-размерность**, 898**VP**

Verb Phrase, 1049

**VPI**

Value of Perfect Information, 800

**W****WAM**

Warren Abstract Machine, 404

**Web-узел**, 68

сопровождающий, 25

**World Wide Web**, 68; 1102; 1110; 1114; 1117; 1140**A****Абстрагирование**, 115

действий, 115

описаний состояний, 115

**Австралия**, 210; 211; 218; 228; 513**Автомат**, 1268; 1273

AFSM, 1226

конечный, 1050

конечный дополненный, 1226; 1228

ситуационный, 1240

- шахматный, 271
- Автомобиль**  
гоночный, 1285
- Автономность**, 82; 513
- Автопилот**, 433
- Агент**, 39; 103  
автономный, 286  
в мире вампуса, 622; 1142  
водитель такси, 102; 1281  
гибридный, 332  
действующий на основе теории решений, 778  
действующий на основе цели, 98; 104  
действующий с учетом полезности, 99; 104; 836; 1011  
для мира вампуса, 288; 363  
ждадный, 1021  
интеллектуальный, 24; 25; 71; 75; 1269  
логический, 284; 376  
на основе знаний, 282  
на основе логического вывода, 333  
на основе логической схемы, 325; 333  
на основе цели, 105  
наивный, 1053  
непрерывно планирующий, 600  
обучающийся, 105  
обучающийся активный, 1020  
обучающийся пассивный, 1012  
общающийся, 1053  
основанный на знаниях, 50; 341; 1277  
основанный на модели, 96  
перепланирующий, 594; 611  
программный, 85  
рациональный, 39; 75; 78; 104; 808; 1277  
рациональный ограниченный, 469  
реагирующий на текущую ситуацию, 1255  
рефлексный, 93; 96; 103; 818; 1012  
решающий задачи, 110
- Агент-пылесос**, 77; 78
- Агентство**  
DARPA, 70; 773  
NASA, 69; 432; 501; 552; 613; 712; 1189  
космическое европейское, 613
- Агрегирование**, 569
- Адалины**, 59
- Адамс Дж.**, 465
- Азимов А.**, 1237
- Аксиома**, 359  
вероятности, 1293  
возможности, 453  
декомпонуемости, 782  
исключения действия, 548  
китайской комнаты, 1264  
мира вампуса, 363  
окружения, 455; 457  
Пеано, 361; 375; 394  
полезная, 424  
полезности, 783
- предусловия, 548  
проблемной области, 442  
результата, 453  
сituационного исчисления, 453  
состояния-преемника, 337; 455; 497; 547  
теории множеств, 362  
теории натуральных чисел, 361  
теории полезности, 782  
универсального действия, 457  
универсальных имен, 457
- Аксиоматизация**  
равенства, 420
- Аксиомы**  
Strips, 557  
вероятностей, 637  
Колмогорова, 635
- Аксон**, 48
- Акт**  
речевой, 1047; 1092; 1093  
речевой декларативный, 1047  
речевой непрямой, 1047
- Алгебра**  
Робинса, 427
- Алгоритм**  
“ожидания–максимизации”, 962  
AC-3, 221  
Boxes, 1033  
CSP, 217  
DPLL, 316  
EKF, 1202  
EM, 962; 964; 966; 968; 969  
EM структурный, 972  
LRTA\*, 204  
MCL, 1199  
MGSS\*, 274  
Pegasus, 1036  
POP, 528  
Q-обучения, 1190  
RBFS, 163  
Reinforce, 1036  
SSS\*, 274  
Unify, 386  
венгерский, 1175  
Витерби, 768  
внутренний–внешний, 1109  
выравнивания, 1177  
генетический, 182; 1023  
диаграммного синтаксического анализатора, 1060  
Дэвиса–Патнем, 316  
заполнения мозаики, 991  
итерации по значениям, 824  
итерации по стратегиям, 830; 1012  
локализации Монте-Карло, 1200  
локального поиска, 318  
Метрополиса, 202; 709  
минимакса, 586  
минимаксный, 245; 845

- Монте-Карло, 683  
 Монте-Карло на основе цепи Маркова, 690; 709  
 неинформированный, 110  
 непротиворечивый, 301; 393  
 обратного логического вывода, 315; 399  
 обратного распространения, 988; 999  
 обучающий слабый, 886  
 обучения с подкреплением, 1028  
 ожидания–максимизация, 1109  
 оптимально эффективный, 162  
 оптимального повреждения мозга, 991  
 перебора с возвратами, 1185  
 поиска в графе, 138  
 поиска в дереве общий, 125  
 поиска общий, 144  
 поиска с наименьшим вкладом, 909  
 полный, 301; 393  
 прямого логического вывода, 313; 392  
 резолюции, 309; 409  
 с отсечением по времени, 1282  
 силуэта, 1239  
 симплексный, 202  
 согласования строк Бойера–Мура, 427  
 сортировки, 438  
 унификаций, 387  
 управляемого конфликтами обратного перехода, 225  
 усиления, 887  
 фильтрации, 836  
 шифрования RSA, 427  
 эмуляции отжига, 694
- Аллен В., 765  
 Алфавит  
     фонетический, 759  
 Альберти Л.Б., 1181  
 Альтруизм, 627  
 Альфа-бета-отсечение, 247; 277  
 Альфа-бета-поиск, 270; 272  
 Альхазен, 1181  
 аль-Хорезми, 43  
 Анализ, 1052  
     алгоритмов, 1288  
     асимптотический, 1289  
     байесовский, 45  
     обесценивания, 856  
     проблемы принятия решений, 803  
     синтаксический, 1052; 1056  
     синтаксический восходящий, 1056  
     синтаксический нисходящий, 1056  
     скрытых закономерностей в данных, 67  
     сложности, 1290  
     сложности алгоритма, 128  
     содержимого узкий, 1260  
     содержимого широкий, 1260  
     чувствительности, 807
- Анализатор
- синтаксический диаграммный, 1059  
 синтаксический по левому углу, 1063
- Аномалия  
 Зюссмана, 555; 560
- Ансамбль  
 гипотез, 885
- Антецедент, 295
- Аппарат  
 космический Deep Space One, 105; 552; 613  
 космический Remote Agent, 428  
 опорно-двигательный, 91
- Аппроксиматор  
 функции, 1030
- Аппроксимация  
 функциональная, 1028
- Аристотель, 38; 41; 42; 47; 104; 105; 333; 374; 429; 496; 499; 1181; 1272
- Арно А., 43; 778
- Арность, 350; 394
- Артефакт, 35
- Архитектура, 91  
 ACT\*, 940  
 Soar, 431  
 агента, 68  
 гибридная, 1227; 1281  
 классной доски, 773  
 когнитивная, 398  
 компьютерная параллельная, 204  
 обобщающая, 1227  
 обобщения, 615  
 программного обеспечения, 1227  
 производственной системы, 398  
 рефлексивная, 1282  
 средство распознавания речи, 67  
 трехуровневая, 1229  
 управления в реальном времени, 433
- Ассистент  
 математика, 107
- Астроном, 713  
 Атанасов Дж., 51  
 Атрибут, 870  
     последовательности состояний, 820
- Аукцион  
 английский, 853  
 Викри, 854  
 с запечатанными предложениями, 853
- Ацикличность, 329
- Б**
- База  
 Эрбрана, 418
- База данных  
 дедуктивная, 429; 432  
 с непререкающимися шаблонами, 173  
 с шаблонами, 172  
 шаблонов, 201

- База знаний, 284; 332  
 логическая, 376  
 хорновская, 423  
 Байес Т., 45; 644; 654  
 Бандит  
 н-рукий, 1022  
 Банк  
 деревьев, 1135  
 Бартлетт Ф., 50  
 Башня  
 Пизанская, 102  
 Беззель М., 146  
 Бекон Ф., 41  
 Беллман Р., 824  
 Бергер Г., 47  
 Бернули Д., 812  
 Бернули Дж., 45; 654  
 Берри К., 51  
 Бета-распределение, 957  
 Библиотека  
 планов, 572  
 Бигелоу Дж., 53  
 Биоинформатика, 1135  
 Биометрия, 1168  
 Бит, 877  
 Бихевиоризм, 49; 53; 105  
 Блокирование, 426  
 Бонапарт Н., 271  
 Брандейс Л., 1268  
 Братья Райт, 37  
 Брахмагупта, 233  
 Бридж  
 карточная игра, 74  
 Броха П., 47  
 Брунеллески Ф., 1181  
 Буль Дж., 43; 375  
 Бэбидж Ч., 52; 271  
 Бэтмен, 616  
 Бьюкенен Б., 62
- B**
- Валрас Л., 45  
 Вампус, 286  
 Вашингтон Дж., 465  
 Вебер Й., 1156  
 Вездеход  
 марсианский автономный, 107  
 Вейценбаум Дж., 1268  
 Вектор  
 поддерживающий, 993; 994  
 Вёлер Ф., 1257  
 Венский кружок, 42  
 Вероятность  
 апостериорная, 626; 632  
 априорная, 626; 630; 653
- безусловная, 626; 630  
 класса, 900  
 конъюнктивная, 665  
 маргинальная, 639  
 переходная, 691  
 условная, 626; 632; 653; 665
- Версия  
 поднятая, 385  
 поднятая правила резолюции, 412  
 Верхеймер М., 1182  
 Вершина, 1059  
 детерминированная, 670  
 трехгранная, 1166  
 утечки, 671  
 Вещество, 449  
 временное, 463  
 нитроароматическое, 938  
 подслащающее искусственное, 1257  
 пространственное, 463
- Взвешивание  
 динамическое, 200  
 с учетом правдоподобия, 687; 709; 754
- Взгляды  
 дуалистические на человеческий разум, 1272
- Взрыв  
 комбинаторный, 62  
 Взятый без кавычек, 469  
 Вивер У., 1137  
 Видеокамера, 1192  
 CCD, 1186  
 Виленский Р., 1264  
 Винер Н., 53; 241; 1137  
 Вино  
 Шато Латур, 1257  
 Виноград Т., 64  
 Витгенштейн Л., 42  
 Вклад  
 наименьший, 527  
 онтологический, 346; 347; 374; 625; 700  
 эпистемологический, 346; 347; 374
- Влияние  
 причинное, 667
- Вложение  
 процедурное, 481
- Возможности  
 нейронной сети, 55
- Вознаграждение, 103; 817; 866; 1010  
 аддитивное, 820  
 обесцениваемое, 821  
 отрицательное, 817  
 среднее, 822
- Возражение  
 противника, 192
- Возрождение  
 итальянское, 653
- Война  
 Вторая мировая, 46; 771

- Вокансон Ж., 1237  
 Воля  
     свободная, 41  
 Вопрос  
     искусственного интеллекта  
         фундаментальный, 1027  
         могут ли машины мыслить, 1249  
 Восприятие, 75; 76; 1051  
     активное, 1141  
     зрительное, 38  
     робототехническое, 1195  
     тактильное, 1141  
     трехцветное, 1148  
 Врач  
     санитарный, 264  
 Вращение  
     трехмерное, 1155  
 Время  
     ожидания ответа, 1115  
 Всеведение  
     логическое, 469  
 Всезнание, 80  
 Вулси К., 1032  
 Вундт В., 49; 1181  
 Выбор  
     места размещения аэропорта, 790  
     площадки для строительства аэропорта, 795  
 Выборка, 683  
     изображений, 1168  
     с исключением, 686  
 Вывод  
     вероятностный, 638; 660  
     грамматики индуктивный, 1089; 1096  
     логический, 284; 292; 380  
     логический непротиворечивый, 332  
     логический обратный, 399; 432  
     логический прямой, 335; 390; 392; 431  
     логический прямой инкрементный, 397  
     логический чисто индуктивный, 867  
     от фактов к цели, 335  
     символический вероятностный, 708  
     формы на основе оттенения, 1183  
     формы на основе текстуры, 1183  
 Выделение  
     основы, 1116  
 Выигрыш  
     наилучший возможный, 789  
 Выметание  
     с учетом приоритетов, 1019  
 Выполнение, 112  
     действий правильное, 35  
     правильных действий, 40; 1283  
 Выполнимость, 302  
 Выработка, 1051  
 Выражение, 307  
     единичное, 307; 317; 421  
     импликационное, 430  
     определенное, 312; 390  
     пустое, 310  
     регулярное, 1121  
     с указанием времени, 1140  
     хорновское, 312; 423; 931  
 Высказывание, 284; 628  
     атомарное, 295; 351; 357; 374  
     в базе знаний, 332  
     диахронное, 364  
     истинное, 352  
     как элемент физической конфигурации агента, 294  
     контрапозитивное, 426  
     неявное, 376  
     с кванторами, 374  
     с оценкой знаний, 328  
     синхронное, 364  
     сложное, 295; 352; 374  
     явное, 376  
 Вычисления  
     нейронные, 977  
 Вычислимость, 44  
 Выявление  
     противоречия, 303
- Г**
- Габор За За, 811  
 Галилей Г., 34; 102; 938  
 Гаусс К.Ф., 146  
 Гёдель К., 44; 409; 430  
 Гексапод, 1225  
 Гелернтер Г., 57  
 Гельмгольц Г., 1181  
 Генезерет М.Р., 286; 1244  
 Генератор  
     вариантов среды, 90  
     проблем, 101  
 Гессиан, 189  
 Гефест, 1237  
 Гештальт-психология, 1182  
 Гильберт Д., 43  
 Гиперграф  
     ограниченный, 213  
 Гиперпараметр, 957  
 Гипотеза, 867  
     максимальная апостериорная, 948  
     приблизительно правильная, 890  
     приблизительно правильная с определенной вероятностью, 889  
     Сапира-Уорфа, 344  
     текущая наилучшая, 897; 905  
     физической символической системы, 57  
 Гирокоп, 1192  
 Гистограмма  
     векторного поля, 1240  
 Глаз, 1141; 1145; 1146; 1181

- Глаза  
  мухи, 1178
- Глубина, 124  
  резкости пространственного изображения, 1145
- Гоббс Т., 41
- Голова  
  выражения, 312; 400
- Головоломка  
  игра в восемь, 170  
  криптоарифметическая, 213  
  с зеброй, 238
- Голосование  
  мажоритарное, 875; 901
- Голубь, 50
- Гольджи К., 47
- Горизонт, 1144  
  бесконечный, 819  
  конечный, 819
- Город  
  Вена, 1259
- Государство  
  Ноуноу, 391
- Гофер, 1232
- Градиент  
  ландшафта, 188  
  стратегии, 1035  
  текстуры, 1162; 1182  
  целевой функции, 188  
  эмпирический, 188; 1035
- Грамматика, 1048; 1297  
  PCFG лексикализованная, 1108  
  арифметических выражений расширенная, 1072  
  атрибутов, 1093  
  вероятностная контекстно-свободная, 1106  
  вероятностная лексикализованная, 1136  
  зависимостей, 1094  
  категориальная, 1094  
  контекстно-зависимая, 1049  
  контекстно-свободная, 1050  
  контекстно-свободная вероятностная, 1135  
  лексически-функциональная, 1094  
  метаморфоз, 1093  
  определенных выражений, 1067; 1093  
  регулярная, 1050  
  рекурсивно перечислимая, 1049  
  системная, 1080  
  универсальная, 1096  
  формальная, 1054
- Граф  
  AND-OR, 314  
  видимости, 1239  
  двуходольный взвешенный, 1175  
  ограничений, 210; 229  
  ориентированный ациклический, 661; 707  
  планирования, 536; 554
- планирования выровненный, 538  
планирования последовательный, 540  
строго k-совместимый, 222
- Эйлера, 204  
экзистенциальный, 478
- График  
  счастливый, 879
- Графика  
  компьютерная, 1142
- Греция, 333
- Грин К., 58
- Группа  
  базовая, 1123
- Гюйгенс Х., 858
- Д**
- да Винчи Л., 41; 1181
- Дальномер, 1191  
  лазерный, 1191
- Данные  
  недостающие, 883
- Дарвин Ч., 185
- Датчик, 75; 84; 1141; 1188  
  активный, 1190  
  вращающегося момента, 1192  
  изображения, 1192  
  инерционный, 1192  
  мусора локальный, 143  
  пассивный, 1190  
  положения, 143  
  проприоцептивный, 1192  
  расстояния, 1191  
  усилия, 1192
- Движение, 1156  
  в сторону, 178  
  кажущееся, 1156  
  охраняемое, 1217  
  позиционирующее, 1207  
  приспособляемое, 1217  
  ситуационное, 68  
  согласующее, 1207
- де Маркен К., 1100
- де Финетти Б., 637
- Деградация  
  трагическая общих пастбищ, 852
- Дедукция  
  натуральная, 430
- Дейкстра Э., 1273
- Действие, 75; 144; 840  
  по восприятию, 582; 799  
  по сбору информации, 1216  
  по сбору информации с помощью датчиков, 592  
  применимое, 516  
  примитивное, 571  
  рациональное, 42  
  релевантное, 523

- рефлекторное, 39  
совместимое, 524  
совместное, 607  
сохраняющее, 537  
условное, 614
- Декарт Р.**, 41; 1181; 1272
- Декодер**  
A\*, 768
- Декодирование**  
стека, 1135
- Декомпозиция**  
действий рекурсивная, 578  
действия, 571  
древовидная, 231  
иерархическая, 570  
исчерпывающая, 445  
плана, 617  
ячеек, 1210  
ячеек точная, 1212
- Декомпонуемость**  
лотереи, 783
- Дельта-правило**, 1029
- Дельфин**, 47; 1046
- Демодулятор**, 424; 438
- Демодуляция**, 420; 434
- Дендрит**, 48
- Деннет Д.**, 1257
- День**  
зимний, 1256  
летний, 1256
- Дерево**  
генеалогическое, 378; 929  
доказательства, 393  
категорий, 1117  
поиска, 122; 123  
регрессии, 884  
решений, 809; 870; 872; 896  
решений одноузловое, 888  
связующее минимальное, 201; 206  
синтаксического анализа, 1052  
соединения, 682  
упакованное, 1100
- Детектор**  
края Кэнни, 997
- Дефект**  
плана, 603
- Дешифратор**  
угла поворота вала, 1192
- Деятельность**  
рациональная, 71  
творческая, 54
- Джевонс У.С.**, 334
- Джеймс Г.**, 50
- Джеймс У.**, 50
- Джефферсон Дж.**, 1256
- Джоуль Дж.**, 938
- Диагноз**  
медицинский, 70; 366; 798; 1269
- Диагностика**, 623  
заболеваний зубов, 623  
медицинская, 63; 655; 668; 702; 709
- Диаграмма**, 1059  
Вороного, 1239  
решения бинарная, 557
- Диаметр**  
пространства состояний, 132
- Дизьюнкт**, 295
- Дизьюнкция**, 295  
без дизьюнктов, 310
- Дилемма**  
заключенного, 841
- Диофант**, 233
- Директива**, 1047
- Дискретизация**, 187; 672
- Дифференцирование**, 917  
символическое, 438
- Длина**  
волны, 1147  
минимальная описания, 898
- Довод**  
исходящий из неспособности, 1250  
исходящий из неформализуемости, 1253
- Доказательство**, 305  
математических теорем автоматическое, 434  
математической теоремы, 60; 74  
неконструктивное, 415  
полноты резолюции, 416  
теорем, 554; 615  
теорем автоматическое, 423
- Документ**  
релевантный, 1111
- Доминирование**, 477  
слабое, 842  
стохастическое, 791; 808  
строгое, 791; 842  
эвристики, 169
- Домоводство**, 79
- Дополнение**, 484  
глагола, 1069  
Кларка, 485; 501  
отглагольное, 1071
- Допустимость**, 302
- Дреббель К.**, 52
- Дуализм**, 41; 1272
- Дуга**  
совместимая, 220
- Душа**, 1272
- Дюрер А.**, 1181
- Е**
- Евклид, 43; 1181
- Европа, 64

Единица  
QALY, 809  
Единорог, 339

**Ж**

Жаккард Ж., 52  
Жишинг Ч., 267  
Жук, 344  
навозный, 81; 106  
Журнал  
по искусственному интеллекту, 72

**З**

Зависимость  
функциональная, 923; 941  
Задача, 113; 144  
“Дары волхвов”, 577  
3SAT, 212; 221  
CSP, 221  
CSP бинарная, 213  
CSP булева, 212  
MDP, 857; 1216  
MDP частично наблюдаемая, 1216  
NP-полная, 145; 926; 1003  
SLAM, 1203  
TSP, 201  
выбора площадки для размещения аэропорта, 812  
игры в восемь, 117; 145; 147; 149; 167; 170; 199  
игры в пятнадцать, 145; 199  
коммивояжера, 121; 146; 201; 206  
коммутативная, 215  
многоатрибутной теории полезности, 820  
недостаточно ограниченная, 320  
неразрешимая, 44  
обратного маятника, 1032  
обучения нереализуемая, 869  
обучения реализуемая, 869  
одновременной локализации и составления карты, 1203  
оптимизации, 175  
ослабленная, 170; 171; 525; 526; 536  
оценки стратегии, 1012  
планирования обхода, 120  
планирования пути, 1207  
последовательного принятия решений, 815; 856  
похищения, 1197  
раскрашивания карты четырьмя цветами, 234  
реальная, 116  
решения кроссворда, 87  
робототехнической сборки, 1217  
с п ферзями, 320  
с п-руким бандитом, 1021  
с бесконечным горизонтом, 856  
с восемью ферзями, 118; 145; 176  
с запасным колесом, 519

с лабиринтом, 191  
с миллионом ферзей, 228; 235  
с миссионерами и каннибалами, 145; 148; 495  
с обезьяней и бананами, 148; 559  
с полковником Уэстом, 391  
с рестораном, 870  
с тремя заключенными, 658  
смены колеса со стертым покрышкой, 519  
со скользящими фишками, 118  
удовлетворения ограничений, 209; 408  
упорядочения сборки, 121  
управления навигацией робота, 121  
упрощенная, 116  
фильтрации, 833; 1196  
частично декомпонуемая, 514

**Заключение**  
импликации, 295

**Закон**  
Мура, 1271  
мышления, 39  
Мэрфи, 108; 141; 152  
общественный, 608

**Заменяемость**  
лотерей, 783

**Замыкание**  
резолюционное, 311; 419

**Запоминание**, 407; 918

**Запрос**, 1110  
конъюнктивный, 432  
логический, 358

**Затенение**, 1156

**Зверь**  
Хопкинса, 1238

**Землетрясение**, 662

**Зима**  
искусственного интеллекта, 65

**Знак**, 1046

**Знание**  
исходный код, 773

**Знания**  
априорные, 867; 902; 915; 927  
диагностические, 645  
общие, 58  
основанные на модели, 645  
предварительные, 81  
фоновые, 284; 414; 914  
фоновые обыденные, 1255

**Знания и действия**, 42

**Значение**, 210  
заданное по умолчанию, 481  
ожидаемое, 259  
ожидаемое денежное, 785  
ожидаемое минимаксное, 260; 275  
полезности нормализованное, 789

**Зог**, 914

**Золото**, 286

**Зона**

- Брока, 47
- Зрение
- машинное. см. Система технического зрения
  - низкого уровня, 1143
- И**
- Иголка в стоге сена, 293
- Игра, 46; 240
- “Дипломатия”, 246
  - Almanac Game, 810
  - Minesweeper, 335
  - Qubic, 266
  - административная, 861
  - азартная, 45
  - бридж, 263
  - в двадцать четыре, 118
  - в крестики-нолики, 242; 271; 276
  - в нарды, 266
  - в пятнадцать, 118
  - в чет и нечет на двух пальцах, 839
  - гекс, 275
  - го, 267
  - гомоку, 266
  - калах, 272
  - карточная, 262
  - координационная, 844
  - Мельница, 266
  - минный тральщик, 335
  - осмотр, 840
  - Отелло, 241; 266
  - повторяющаяся, 848
  - покер, 656
  - против природы, 585
  - реверси, 241; 266
  - роботизированная с людьми, 1245
  - с жеребьевкой, 262
  - с ненулевой суммой, 279
  - с нулевой суммой, 844
  - с постоянной суммой, 844
  - с частичной информацией, 850
  - со ставками, 638
- Игрок, 840
- Игрушка
- Radio Rex, 773
- Идентификация
- в пределе, 895; 897
- Иерархия
- обобщения, 912
  - таксономическая, 65; 444
- Избегание
- риска, 787
- Извлечение
- информации, 1102; 1121; 1134
  - характеристик, 1141
- Изменение
- непрерывное, 463
- ИИ
- искусственный интеллект, 24
- ИЛИ-параллелизм, 406
- Имитатор
- среды общего назначения, 90
- Импликация, 295
- двусторонняя, 295
- Имя
- длинное составное, 371
- Индекс
- Гиттинса, 1023; 1040
  - инвертированный, 1119
- Индексальный, 1094
- Индексация
- по второму параметру, 389
  - по предикатам, 388
  - фактов в базе знаний, 388
- Индивидуализация, 449
- Индивидуум, 182
- Индия, 54; 233
- Индукция, 867
- конструктивная, 930
  - математическая, 44
- Инженер
- по знаниям, 367
- Инженерия
- знаний, 367; 441; 665; 1108
  - знаний для экспертных систем, 804
  - онтологическая, 441
- Институт
- MIT, 1239
  - SRI, 555; 809
  - Массачусетский технологический, 55; 56
- Интегрирование
- символическое, 913
- Интегрируемость, 1163
- Интеллект
- искусственный, 24; 56
  - искусственный добрый старый, 1253
  - искусственный реального времени, 1282
  - искусственный сильный, 1248
  - искусственный слабый, 1248
- Интеллектуальность, 35
- Интервал
- времени, 461; 505
  - временной, 498
  - защиты, 530
- Интерпретатор
- Prolog, 404
- Интерпретация, 350
- вероятностная нейронной сети, 984
  - намеченная, 350
  - прагматическая, 1052
  - расширенная, 353
  - семантическая, 1052; 1094; 1098
- Интроспекция, 37

- И-параллелизм, 406  
 Исключение, 441  
     взаимное, 538  
     из суммы, 639  
     путем суммирования, 680  
 Искусственный интеллект, 34  
     направления, 34  
     определение, 35  
     проектирование рациональных агентов, 39  
     универсальная научная область, 34  
**Использование**  
     совместное подзадач, 575  
**Исправление**  
     орфографических ошибок, 1137  
**Исследование**  
     операций, 46; 104; 146; 200; 613  
     ситуации, 81  
     среды, 1012; 1021  
**Истинность**, 290  
**История**  
     пребывания в среде, 817  
**Ичисление**, 187  
     вариационное, 202  
     гедонистическое, 808  
     ситуационное, 451; 459; 497  
     событий, 459; 498; 1074  
     флюентных высказываний, 457; 463; 497  
**Итерация**  
     по значениям, 822; 856  
     по стратегиям, 829; 856  
     по стратегиям асинхронная, 831  
     по стратегиям модифицированная, 831  
**ИТЕФ**  
     институт теоретической и  
     экспериментальной физики, 272
- K**
- Калман Р., 738  
 Камера-обскура, 1144  
 Кардано Дж., 45; 653  
 Карнап Р., 42; 635; 654  
 Карта  
     дорожная вероятностная, 1215  
     коэффициентов отражения, 1163  
     Румынии, 114  
 Каспаров Г., 69; 264  
 Категория, 443  
     события, 505  
 Каҳал С., 47  
 Качество, 1260; 1262; 1274  
 Квантование  
     векторное, 761  
 Квантор, 352; 375  
     всеобщности, 352; 381  
     существования, 354  
 Кванторы  
     вложенные, 355  
 Кеведо Л., 271  
 Кей М., 1125  
 Кемпе А., 1253  
 Кент К., 467  
 Кеплер Й., 1181  
 Кибернетика, 52; 53  
 Килиманджаро, 1046  
 Кинематика, 1209  
     обратная, 1209  
**Кинофильм**  
     “Attack of the Cat People”, 1082  
     “Take the Money and Run”, 765  
**Класс**  
     вариантов среди, 90  
     задач PSPACE, 1291  
     закрытый, 1054  
     открытый, 1054  
     референтный, 634; 655  
**Классификация**, 896; 1117  
     в описательной логике, 482  
     документов, 1117  
**Кластеризация**, 682; 707  
     агломеративная, 1118  
     документов, 1117  
     неконтролируемая, 962; 964  
**Клинтон У.Дж.**, 1168  
**Клуб**  
     Разума, 1002  
**Ключ**  
     хэш-таблицы комбинированный, 389  
**Книга**  
     “Port-Royal Logic”, 778; 808  
     Principia Mathematica, 56  
**Ко-NP-полный**, 301; 335  
**Коартикуляция**, 764  
**Ковальский Р.**, 401  
**Когнитология**, 37; 51; 72  
**Код**  
     открытый, 404  
**Количество**  
     оставшихся значений минимальное, 217  
**Колледж**  
     Дартмутский, 55  
**Коммутативность**, 215  
**Комната**  
     китайская, 1275  
**Компания**  
     Du Pont, 65  
     General Motors, 1237  
     Hitachi, 580  
     IBM, 56; 57; 69; 264; 265; 773  
     Jaguar Cars, 612  
     Microsoft, 711  
     Price Waterhouse, 612  
     Sun Microsystems, 1268  
     Westinghouse, 612

- Херох, 1125  
Компиляция, 405  
Композициональность, 342  
Композиция  
    подстановок, 401  
Компонент  
    обучающий, 100  
    производительный, 100; 102  
    смешанного распределения, 963  
Компоновка  
    СБИС, 121; 146; 180  
    ячеек, 121  
Компрометация  
    проверочных данных, 880  
Компьютер, 51  
    ABC, 51  
    Colossus, 51  
    ENIAC, 51  
    IBM 704, 266  
    NavLab, 69  
    Snarc, 55  
    VAX, 431  
    Z-3, 51  
    вычислительные ресурсы, 49  
Конкретизация  
    высказывания с квантором всеобщности, 381  
    высказывания с квантором существования, 382  
Конкуренция, 610  
Коннекционизм, 66; 977  
Консеквент, 295  
Константа  
    сколемовская, 382; 430  
Консультация  
    юридическая, 74  
Контекст, 343  
    формы, 1174  
    целостный, 1254  
Контроллер, 104; 1221  
    опорный, 1221  
    оптимальный, 1221  
    строго стабильный, 1222  
Контроль  
    выполнения, 582; 594; 611; 615  
    действий, 594; 595; 597  
    плана, 594; 597  
Контур, 1156  
    равных f-стоимостей в пространстве состояний, 160  
Конъюнкт, 295  
Конъюнкция, 295  
Кора  
    головного мозга, 48  
Корень  
    квадратный, 93  
Корпорация  
    Honda, 1189  
Корпус, 1195  
Коррекция  
    орфографических ошибок, 1116  
Корреляция  
    взаимная, 1157  
Кортеж, 348  
Коэффициент  
    ветвлений, 126; 168; 921  
    ветвлений эффективный, 168; 200; 251  
    квантования, 760  
    конкурентоспособности, 191  
    обесценивания, 821; 1014  
    приращения, 884  
    уверенности, 64  
    усилений контроллера, 1222  
Край, 1150; 1165  
Крамник В., 265  
Краткость, 330  
Кривая  
    ROC, 1115  
    обучения, 879; 896; 989  
Критерий  
    коэффициента приращения, 901  
Критик  
    обучающегося агента, 101  
Кроссворд, 70  
Крыса, 50  
Круг К.Дж., 50  
Ктесибий из Александрии, 52  
Кубик  
    Рубика, 145; 171; 1156  
Культ  
    компьютероцентризма, 1249  
Кэнни Дж., 1153

## Л

- Лаборатория  
    ATT Bell Labs, 773  
Лавлейс А., 52  
Лампа  
    электронная, 55  
Ландшафт  
    пространства состояний, 175  
Лаплас П., 45; 654; 699  
Ларсон Г., 914  
Лёвенхейм Л., 375  
Ледерберг Дж., 63  
Лейбниц Г.В., 41; 187; 334; 654; 858  
Лексикон, 1119  
Лемма  
    поднятия, 417; 419  
    Робинсона, 419  
Ленат Д.Б., 507  
Лес  
    упакованный, 1065  
Лимб, 1165

- Лингвистика, 72  
 вычислительная, 54  
 компьютерная, 72
- Линеаризация, 1200  
 плана, 528
- Линза, 1145
- Линии  
 параллельные, 1144
- Линия  
 базисная, 1161  
 Вороного, 1214  
 задержки, 326
- Линней К., 496
- Литерал, 295; 515  
 базовый, 515  
 не содержащий функций, 515  
 ответа, 415  
 отрицаемый, 425  
 отрицательный, 295; 515  
 положительный, 295; 515
- Литералы  
 взаимно обратные, 307
- Лицо  
 анализирующее решения, 804  
 принимающее решения, 804
- Лloyd С., 145
- Ловушка  
 предсказуемости, 88
- Логика, 38  
 булева, 294; 334  
 временная, 346; 496; 497  
 высокого порядка, 346  
 динамическая, 497  
 индуктивная, 635; 655  
 модальная, 468; 499  
 немонотонная, 489; 501  
 нечеткая, 625; 700; 711  
 описательная, 477; 482; 495; 500  
 первого порядка, 283; 341; 345  
 пропозициональная, 283; 294; 342  
 система обозначений, 38  
 умолчания, 490; 501  
 формальная, 43
- Логит-распределение, 674; 708
- Логицизм, 38
- Логическое  
 программирование в ограничениях, 235
- Лодж Д., 1285
- Лозунг  
 "Мозг рождает разум", 1259
- Локализация, 1197  
 глобальная, 1197  
 марковская, 1238  
 Монте-Карло, 1199
- Локальность, 700
- Локатор  
 звуковой, 1191
- Локк Дж., 41
- Лотерея, 781; 811  
 стандартная, 789
- Лошадь, 1259
- Луллий Р., 41
- Любовь, 1250
- Любопытство, 1024
- М**
- Макаллестер Д.А., 66
- Макдональд Р., 345
- Мак-Каллок У., 53
- Маккарти Дж., 55; 57; 1264
- Макнили С., 1268
- Макрооперация, 613
- Максвелл Дж., 699; 1181
- Максимин, 844
- Максимум  
 локальный, 177  
 нормализованный, 827
- Малик Дж., 1156
- Мандела Н., 1168
- Манипулятор  
 stanfordский, 1193
- Маргинализация, 639
- Мартышка-верветка, 1046
- Маршрутизация  
 каналов, 121
- Маслов С., 431
- Масс-спектрометр, 63
- Мат, 136
- Математик  
 древнегреческий, 429  
 советский, 635
- Математика, 43; 56
- Материализм, 41; 1273  
 элиминирующий, 1274
- Матрица  
 вознаграждений, 840  
 переходов, 716
- Махавиракарья, 653
- Машина  
 Demonstrator Стенхупа, 334  
 WAM, 433  
 аналитическая, 52  
 Болтымана, 1004  
 обучающаяся, 100  
 поддерживающих векторов, 895; 898; 991; 997; 1004  
 поддерживающих векторов виртуальная, 997  
 разностная, 52  
 рука-глаз, 1239  
 сверхинтеллектуальная, 1270  
 Тьюринга, 44; 898  
 Уоррена абстрактная, 404; 433
- Мегапеременная, 734

- Мендель Г., 185  
 Менингит, 658  
 Мера, 448  
 Мереология, 498  
 Метаданные, 1116  
 Метаправило, 409  
 Метарассуждения  
     на основе теории решений, 1282  
 Метафизика, 42  
 Метафора, 1083; 1095  
 Метка  
     в плане, 588; 619  
     линии, 1165  
 Метод  
     ЕКФ, 1205  
     SLD-результаты, 432  
     безмодельный, 1026  
     временной разности, 1017  
     инверсный, 431  
     итерационный, 824  
     кластеризации по к средним, 1118  
     критического пути, 566  
     локализации Монте-Карло, 1201  
     магических множеств, 432  
     Монте-Карло на основе цепи Маркова, 706; 754  
     надежный, 1217  
     непосредственной оценки полезности, 1014  
     Ньютона-Рафсона, 188  
     обнаружения краев Кэнни, 1183  
     обратного перехода, 224  
     обратного распространения, 1002  
     обратного распространения ошибки, 987  
     перевода с помощью памяти, 1127  
     передачи, 1127  
     предпочтения единичных выражений, 422  
     результаты, 58; 61  
     скелетирования, 1239  
     слабый, 62  
     соединения, 431  
     усиления, 887; 897  
     устранения моделей, 432  
 Метонимия, 1082; 1095  
 Механизм, 851  
     защиты стратегии, 853  
     исполнительный, 75; 84; 1188  
     поиска общего назначения, 62  
 Механика  
     статистическая, 65; 1002  
 Микромир, 58; 59; 61  
 Микрошанс  
     смерти, 789; 809; 812  
 Милл Дж.С., 43  
 Миллер Дж., 51  
 Минимизация  
     логическая, 447  
 Минский М.Л., 55; 58; 72  
 Мир  
     блоков, 59; 64; 502; 520; 600  
     вампуса, 286; 337; 363; 442; 653; 659; 1047  
     возможный, 290  
     минного тральщика, 339  
     пылесоса, 77; 78; 107; 116; 152; 589  
     пылесоса с альтернативным двойным законом Мэрфи, 593  
     пылесоса с двойным законом Мэрфи, 585  
     пылесоса с тройным законом Мэрфи, 588  
 Миры  
     возможные, 499  
 Мистицизм, 48  
 Мичи Д., 72  
 Множества  
     непересекающиеся, 445  
 Множество  
     в логике первого порядка, 362  
     граничное, 909  
     конфликтное, 224; 225  
     магическое, 399  
     нечеткое, 711  
     обучающее, 873; 879; 896  
     обучающее взвешенное, 885  
     обучающее тиражированное, 885  
     ответов, 488  
     поддержки, 422; 424  
     прроверочное, 879; 896  
     разрыва цикла, 231  
     случайное, 705  
     состояний, 589  
     строк, 1048  
 Модальность  
     сенсорная, 1141  
 Моделирование  
     мира, 1260  
     языковое, 1112  
 Модель, 290  
     HMM, 773  
     IBM Model 3, 1131; 1140  
     n-словных сочетаний, 1103  
     акустическая, 758; 1085  
     байесовская наивная, 1113  
     векторного пространства, 1120; 1136  
     восприятия, 722; 734; 740; 770; 833; 1197  
     графическая, 661  
     движения, 1197  
     двухсловных сочетаний, 766; 1103  
     двухсловных сочетаний вероятностная, 1116  
     ключевых слов булева, 1111  
     марковская скрытая, 67; 718; 746; 771; 772  
     минимальная, 486; 489  
     мира, 96; 1084  
     мыслительная, 1084  
     наблюдения, 832  
     наивная байесовская, 648; 967  
     однословных сочетаний, 1103

- перевода, 1129; 1130  
 перехода, 755; 817; 835; 1013; 1197  
 постоянного отказа, 750  
 причинная, 668  
 прогностическая, 1010  
 реляционная вероятностная, 695  
 скрытой семантической индексации, 1135  
 среды, 1012  
 стохастического движения, 816  
 трехсловных сочетаний, 766; 1103  
 фертильности, 1132  
 ядерная, 975  
 языковая, 758; 1084; 1129; 1132  
 языковая вероятностная, 1103  
 языковая двухсловная, 1103  
 языковая однословная, 1103
- Модуль**  
 Office Assistant, 707  
 Printer Wizard, 707
- Мозг, 54**  
 вычислительные ресурсы, 49
- Мозг и разум, 48**
- Монотонность, 305**  
 предпочтений, 783  
 эвристической функции, 160
- Моргенштерн О., 45**
- Мороженое, 627**
- Мочевина**  
 искусственная, 1257
- Мочли Дж., 51**
- Музей**  
 науки лондонский, 52
- Музыка, 52**
- Мультимножество**  
 слов, 1113
- Мур Т., 55**
- Мураками Т., 266**
- Мутация, 61; 183**
- Муха**  
 летящая, 1158
- Мышление, 46**
- Мышь**  
 летучая, 616
- механическая Theseus, 897**
- H**
- Наблюдения**  
 независимые и одинаково распределенные, 947
- Набор**  
 результирующий, 1111
- Навыки**  
 моторные, 74
- Наказание**  
 вечное, 849
- Намерение, 1050**
- Нарды, 275; 1029**
- Наслаждение**  
 земляникой, 1250
- Наследование, 444; 479; 509**  
 множественное, 479
- Насыщение, 418**
- Наука**  
 прикладная, 72  
 проектирования рациональных агентов, 68  
 теоретическая, 72
- Неврология, 46**  
 вычислительная, 977
- Невычислимость, 44**
- Недетерминированность**  
 неограниченная, 581  
 ограниченная, 581
- Недоказуемость, 44**
- Недопроизводство, 1055**
- Независимость, 642**  
 абсолютная, 642; 653  
 взаимная по полезностям, 795  
 взаимная по предпочтениям, 794  
 маргинальная, 642  
 по предпочтениям, 812  
 подцелей, 525  
 полезностей, 795  
 предпочтений, 793  
 условная, 647; 651; 653; 658; 669; 706; 729
- Незнание, 700; 703**
- Нейробиология, 1184**
- Нейрон, 47; 54; 1262**
- Немонотонность, 489**
- Неоднозначность, 343**  
 лексическая, 1081  
 синтаксическая, 1081  
 структурная, 1081
- Неоднородность**  
 сосредоточенная, 1150
- Неопределенность, 64; 68; 441; 622; 703; 1255**  
 идентичности, 698  
 реляционная, 697  
 управления, 1218
- Неосведомленность, 700**
- Неполнота**  
 системы Prolog, 406
- Непрерывность**  
 предпочтений, 782
- Непрозрачность**  
 ссылочная, 468; 499
- Непротиворечивость**  
 логического вывода, 293
- Неравенство**  
 треугольника, 160
- Неразрешимость, 44; 61**
- Нерациональность, 35; 781**

Никсон Р., 489; 1084

Нили Р., 266

Нильссон Н., 1244

Номенклатура

биноминальная, 496

Номер

гёделевский, 417

телефона, 470

Норвиг П., 449

Нормализация, 640

Нуль-гипотеза, 882

Ньютон И., 34; 93; 187; 723

Ньюэлл А., 37; 51; 55

## О

Обеспечение

аппаратное нейронной сети, 55

Область

действия кванторов, 1095

определения, 210

определения бесконечная, 212

определения конечная, 211; 408

определения непрерывная, 212

определения случайной переменной, 628

проблемная в логике первого порядка, 347

проблемная в представлении знаний, 358

проблемная родства, 358

проблемная семейных отношений, 358

проблемная электронных схем, 369

Обнаружение

краев, 1150; 1151

Обновление

Беллмана, 825

Беллмана упрощенное, 831

Обобщение, 423; 867; 905; 906

в описательной логике, 482

на основе объяснения, 268

Оболочка

для извлечения информации из Web-

страницы, 476

Обоснование, 294; 487

Обработка

естественного языка, 54

изображения, 1180

параллельная распределенная, 977

потока данных, 326

специальных ограничений, 222

текстов на естественных языках, 36

Обусловленность

причинная, 298; 646

Обучение, 24; 81; 286; 864; 1108; 1255

активное, 1012

ансамбля, 884

байесовское, 889; 946; 998

в мире блоков, 59

игре в шашки, 57

индуктивное, 173; 896

индуктивное на основе базы знаний, 927

индуктивное на основе знаний, 917; 939

инкрементное, 909; 914

классификации, 870

контролируемое, 865; 1029; 1169; 1255

кумулятивное, 939

машинное, 36; 39

на опыте, 1250

на основе накопления знаний, 930

на основе объяснения, 915; 939; 940

на основе опыта, 68

на основе релевантности, 939

на основе экземпляра, 972; 1040

нейронной сети, 55

неконтролируемое, 866

непараметрическое, 972

оперативное, 1029

параметрам, 950

параметрам с максимальным

правдоподобием, 953

параметрическое, 972

пассивное, 1012

путем обратного распространения, 62; 66

регрессии, 870

с подкреплением, 866; 1011; 1255

с подкреплением иерархическое, 1041

с учетом релевантности, 916; 924

хеббовское, 55

Общение, 1046

Общество

по искусственному интеллекту, 72

Объект, 345; 351

составной, 446

Объективизм, 634

Объяснение, 493; 702; 919

Овеществление, 444; 467

О'Генри, 577

Ограничение, 210

“самое большое”, 223

Alldiff, 222

бинарное, 213

линейное, 212

логического следствия, 913; 930; 939

на предпочтение, 781

на ресурсы, 567

нелинейное, 212

неравенства, 535

предпочтения, 214

ресурсное, 223

связи, 1208

состояния, 518; 549

унарное, 213

упорядочения, 529; 533

целостности, 312

Одометрия, 1192

Оккам У., 868; 896

Октаант, 1166

- Олдисс Б., 1271  
 Омофон, 758  
 Онтология, 368; 371  
     верхняя, 441; 494  
 Оператор  
     модальный, 468; 496  
 Операция  
     унификации, 430  
     факторизации, 412  
 Описание  
     PEAS, 83; 85  
     косвенное, 489; 501  
     косвенное с приоритетами, 490  
     с помощью переменных, 919; 920  
     снега словесное, 344  
 Оплодотворение  
     искусственное, 1257  
 Определение, 923; 941; 943  
     логическое, 359  
     минимальное, 926  
     ограниченный с помощью обучения, 226  
     потенциальное, 903  
     рационального агента, 79  
     рекурсивное, 933  
     условий выбора множества разрыва  
         цикла, 231  
 Опровержение, 303  
 Оптимальность  
     алгоритма поиска, 126; 144  
     ограниченная, 1284  
     ограниченная асимптотическая, 1285  
 Оптимизатор  
     локальный, 579  
 Оптимизация  
     компилятора, 579  
     с ограничениями, 189  
 Организация  
     IEEE, 496  
     Open Mind Initiative, 496  
     Sigir, 1138  
 Орнитоид, 616  
 Оса-сфекс, 81; 599  
 Основа, 1254  
 Осуществимость  
     логического вывода, 483  
 Осязание, 1141  
 Отбор, 183  
 Ответ  
     системный, 1264  
 Ответственность  
     правовая, 1268  
 Отделение, 701  
 Отжиг, 180  
 Отзывы  
     касающиеся релевантности, 1117  
 Отказ
- датчика, 749; 750  
 Отклик, 50  
 Открытие  
     научное, 897  
 Отладка, 368  
     автоматизированная, 942  
     базы знаний, 373  
 Отметка, 1198  
 Отношение, 345  
     взаимно исключающее, 554  
     взаимного исключения, 554  
     зашумленного OR, 670  
     нейтральное к риску, 787  
     предпочтения стационарное, 820  
     равенства, 420  
 Отношения  
     причинно-следственные, 298; 366  
     связности, 1087  
     связности речи, 1096  
 Отплата  
     “зуб за зуб”, 849  
 Отражение  
     диффузное, 1147  
     зеркальное, 1147  
 Отражения  
     взаимные, 1163  
 Отрицание, 295  
     как недостижение цели, 403  
 Отсечение, 162; 241; 881  
     в задачах с непредвиденными  
         ситуациями, 261  
     ветвей дерева решений, 881  
     ветви, 921  
     ненужных ходов, 265  
     поддерева, 161  
     предварительное, 256  
     хи-квадрат, 882  
 Отслеживание  
     траектории, 1197  
 Отсутствие  
     практических знаний, 624  
     теоретических знаний, 624  
 Отчет  
     ALPAC, 1138  
     Лайтхилла, 62; 65  
     Олви, 65  
     парламентский, 1128; 1140  
 Оценка  
     вероятностная, 634  
     позы, 1171  
     согласованная, 685  
     степени уверенности с интервальным  
         значениями, 700  
     стратегии, 829  
 Очередь, 125  
     FIFO, 127  
     LIFO, 130

по приоритету, 1119  
последовательная, 127

**Ошибка**  
гипотезы, 890

**П**

**Падение**  
монеты, 701

**Память**  
ассоциативная, 1004  
долговременная, 398  
кратковременная, 398  
рабочая, 941

**Панини**, 54

**Пара**  
“переменная–терм”, 358

**Парадокс**, 499  
Никсона, 489  
санкт-петербургский, 808; 812

**Параллакс**  
движения, 1158; 1181

**Параметр**, 672

**Парамодуляция**, 421; 434

**Пари**  
Паскаля, 808

**Паскалина**, 41

**Паскаль Б.**, 41; 45; 653

**Пенроуз Р.**, 1252

**Перевод**  
машинный, 61; 74; 1102; 1124  
машинный статистический, 1127  
принудительный, 583  
текста на естественном языке, 61

**Перезапись**  
термов, 434

**Переименование**, 392

**Перейра Ф.**, 1050

**Перекрытие**  
заданного по умолчанию значения, 481

**Переменная**, 210  
вспомогательная, 213  
запроса, 675  
индикаторная, 964  
латентная, 961  
логическая, 353; 404  
свидетельства, 675  
скрытая, 675  
случайная, 628; 666  
случайная булева, 629  
случайная непрерывная, 672; 708

**Перемещение**  
связок → внутрь выражений, 410

**Перепланирование**, 582; 595; 615

**Перепроизводство**, 1055

**Пересмотр**  
убеждений, 491

**Переход**  
к общим заключениям, 914  
корректный к более примитивному  
поведению, 838  
обратный, 235  
принудительный, 614  
фазовый, 336

**Периферия**, 124

**Перл Дж.**, 662

**Персептрон**, 999; 1002  
простой, 1007  
с голосованием, 1003

**Перспектива**, 1181

**Перфокарта**, 52

**Перцептрон**, 59  
репрезентативные возможности, 62  
с двумя входами, 62

**Петля**  
обратной связи, 702

**ПЗС**  
прибор с зарядовой связью, 1143

**Пивзавод**, 615

**Пикассо П.**, 1257

**Пиквик Г.**, 1256

**Пиксел**, 1144

**Пингвин**, 616

**Пинг-понг**, 74

**Пирс Ч.С.**, 234; 478

**Питтс У.**, 53

**План**  
действий в непредвиденных ситуациях, 145  
нелинейный, 556  
совместный, 606  
согласованный, 530  
универсальный, 616  
условный, 611; 803

**Планетоход**, 1189

**Планирование**, 98; 270; 453; 512  
HTN, 1041  
авиапутешествий, 120  
без использования датчиков, 581  
без чередования, 560  
в мире блоков, 59  
военных операций, 120  
движений тонкое, 1217  
история развития, 554  
классическое, 512  
линейное, 555  
маршрута, 58  
многотельное, 606  
мультиагентное, 605  
на основе прецедентов, 613  
неклассическое, 512  
непрерывное, 582; 600; 601; 615  
операций авианосца, 615  
покрытия пола, 237  
прогрессивное, 522

- производства, 565  
 реактивное, 615  
 регрессивное, 524  
 с помощью иерархической сети задач, 611  
 с регрессией от цели, 555  
 с учетом непредвиденных ситуаций, 582  
 с частичным упорядочением, 528  
 совместимое, 581; 1217  
 согласованное, 614  
 условное, 582; 623
- Планировщик**  
 FastForward, 557  
 PRS, 616  
 Remote Agent, 552
- Плато, 177
- Платон, 333; 498; 1272
- Поведение**  
 стохастическое, 88  
 эмерджентное, 609; 1226
- Поверхность**  
 ламбертова, 1147; 1163; 1185
- Подбрасывание**  
 монеты, 703; 812; 877
- Подгонка**  
 чрезмерно щательная, 881; 990
- Поддержка**  
 несогласованная, 539  
 связи, 88
- Подкрепление, 866; 1010
- Подмножество**  
 английского языка, 1054
- Поднятие, 385
- Подсобытие, 461
- Подстановка, 358; 381  
 Compose, 401
- Подход**  
 "разделяй и властвуй", 552  
 близорукий, 803  
 декларативный, 285; 342  
 логистический, 375  
 процедурный, 286; 342
- Подцели**  
 упорядочиваемые, 552
- Поездка**  
 в аэропорт, 622
- Поза, 1154; 1175; 1193
- Позитивизм**  
 логический, 42
- Позиция**  
 пропозициональная, 467  
 спокойная, 255; 272  
 целенаправленная, 1274
- Поиск, 98; 112; 144  
 A\*, 157  
 A\* с итеративным углублением, 163; 200  
 A\* с ограничением памяти, 165; 201  
 B\*, 274
- LRTA\*, 196  
 в Internet, 122  
 в автономном режиме, 189  
 в глубину, 130; 144  
 в глубину с итеративным углублением, 133  
 в непрерывном пространстве, 202  
 в оперативном режиме, 189  
 в реальном времени, 204  
 в ширину, 127; 144  
 двунаправленный, 135; 136; 144; 201; 560  
 жадный, 176  
 жадный локальный, 176  
 жадный по первому наилучшему совпадению, 155  
 информационный, 1102; 1110; 1136  
 информированный, 127; 153; 154  
 линейный, 188  
 локальный, 153; 174; 235; 320; 335  
 локальный лучевой, 181  
 лучевой, 181  
 маршрута, 120  
 минимаксный, 249; 268; 270  
 неинформированный, 127; 129; 146  
 параллельный, 204  
 паросочетаний, 1175  
 по критерию стоимости, 129; 144  
 по первому наилучшему совпадению, 154; 198  
 при переводе, 1132  
 противоречий, 431  
 рекурсивный по первому наилучшему совпадению, 163; 201  
 с возвратами, 131; 215; 228; 234  
 с возвратами динамический, 235  
 с возвратами хронологический, 224  
 с восхождением к вершине, 175; 194; 207  
 с восхождением к вершине и перезапуском случайнym образом, 179  
 с запретами, 202  
 с итеративным углублением, 133; 144; 146; 255; 425  
 с итеративным удлинением, 135; 149  
 с ограничением глубины, 132; 144  
 с перезапуском случайнym образом, 207  
 с частичной информацией, 139  
 с эмуляцией отжига, 180; 207  
 слепой, 127  
 спокойных позиций, 255  
 стохастический лучевой, 182  
 стохастический с восхождением к вершине, 178  
 стратегии, 1033  
 текущей наилучшей гипотезы, 905  
 эвристический, 127; 199  
 эвристический с ограничением памяти, 200
- Показатели**  
 производительности, 623  
 производительности водителя, 83

- Показатель  
производительности, 78; 79; 780  
связности, 1104
- Покрытие  
марковское, 669; 716
- Поле  
потенциалов, 1213
- Полезность, 99; 241; 817  
ожидаемая, 105; 627; 778; 779; 785  
ожидаемая максимальная, 627; 779; 783; 808  
состояния, 823
- Полет  
искусственный, 37
- Полидерево, 681; 706
- Политика  
оптимальная, 1216
- Полковник  
Уэст, 391
- Полнота, 331  
алгоритма поиска, 126; 144  
выборки, 1114  
доказательства, 293; 409  
опровержения, 308; 416  
процедуры доказательства, 332  
резолюции, 310; 416
- Полуход, 243
- Получение  
высказывания, 293
- Понимание, 1048  
естественного языка, 59; 64  
речи, 758  
языка, 54
- Понятие  
человек, 360
- Понятия  
хорошего или плохого, 779
- Популяция, 182
- Попытка, 1013
- Порфирий, 499
- Последовательность  
актов восприятия, 76; 79  
состояний, 823
- Последствие  
действия, 518
- Последствия  
 побочные, 852
- Поток  
оптический, 1156; 1179; 1182
- Потомок  
вершины в байесовской сети, 669
- Потребление  
полученных результатов, 1021
- Потребность  
в памяти, 128
- Почта  
электронная нежелательная, 1139
- Поэзия, 34
- Правдоподобие  
максимальное, 950; 951
- Правила, 616
- Правило  
“экономии количества ставок”, 783  
if-then, 94  
Modus Ponens, 303  
Байеса, 45; 644; 646; 653; 657; 758  
введение квантора существования, 435  
Видроу–Хоффа, 1029  
вывода в логике первого порядка, 384  
де Моргана, 356  
демодуляции, 420  
Демпстера, 703  
диагностическое, 365  
единичной резолюции, 307  
если–то, 295  
импликации, 295  
леворекурсивное, 1057  
логического вывода, 303; 307; 333  
логического вывода для кванторов, 381  
модус поненс, 303  
обновления, 55  
отделения, 303; 334; 423; 428; 437; 468  
отделения обобщенное, 385; 429  
параметризации, 421  
перезаписи, 424; 438  
подстановки, 1049; 1298  
полной резолюции, 307  
правой ассоциации, 1084  
применимое по умолчанию, 490  
причинное, 366  
произведения, 632  
прохождения, 430  
резолюции, 306; 430  
резолюции для логики первого порядка, 412  
ситуация–действие, 94  
удаления двухсторонней импликации, 304  
удаления связки “И”, 304  
условие–действие, 94; 275  
хеббовского обучения, 55  
цепное, 665
- Правительство  
британское, 62
- Прагматика, 1048
- Превосходство  
технологическое, 1270
- Предел  
глубины, 255
- Предикат, 1072  
append, 405  
assert, 402  
perm, 438  
Person, 360  
retract, 402  
sorted, 438

- ввода-вывода, 402  
целевой, 870
- Предложение**, 1055  
Гёделя, 1251  
констатирующее, 42  
относительное, 1055
- Предположение**, 493  
Strips, 516  
Гольдбаха, 943  
марковское, 771  
о замкнутом мире, 484; 515  
о марковости, 721  
о независимости предпочтений, 820  
об открытом мире, 546  
об уникальности имен, 457; 484
- Предположения**  
исходные, 1156
- Предпосылка**, 295
- Предпочтение**, 626; 783  
моделей, 489  
монотонное, 785
- Предпочтения**  
человека, 821
- Предсказание**  
Саймона, 60
- Представление**, 209  
в виде модели, 50  
времени, 496  
знаний, 36; 54; 58; 64  
знаний аналогическое, 376  
наивное байесовское, 655  
стратегии стохастическое, 1034
- Предусловие**, 515  
внешнее, 573  
открытое, 530; 603
- Предусловия**  
знаний, 470
- Предшественник**  
узла, 136
- Преемник**, 523  
узла, 136
- Преемственность**  
эвристической функции, 160
- Президент**, 465
- Преимущество**  
материальное, 254
- Премия**  
Американской академии киноискусства, 616  
Нобелевская, 46; 47; 63  
Оскар, 616  
страховая, 787  
Тьюринга, 1295  
Фредкина, 273
- Пренебрежение**  
служебными обязанностями, 1269
- Преобразователь**  
каскадный с конечными автоматами, 1123
- Препятствие**  
вертикальное, 1210
- Пресли Э.**, 464
- Приведение**  
к абсурду, 303  
к нижнему регистру, 1116
- Привод**  
гидравлический, 1195  
дифференциальный, 1194  
пневматический, 1195  
синхронный, 1194
- Пригодность**  
операционная, 921
- Приемник**  
GPS, 1191
- Приз**  
Лебнера, 73
- Приложение**, 1071
- Применение**  
эталонных тестов, 1288
- Пример**, 867  
можно отрицательный, 904  
можно положительный, 904  
отрицательный, 870  
положительный, 870
- Принуждение**, 581
- Принцип**  
"сформулировать, найти, выполнить", 112  
"разделяй и властвуй", 774  
безразличия, 634; 654  
бритвы Оккама, 868; 896; 898; 913; 934; 949  
индукции, 42  
модифицированной итерации по  
стратегиям, 1015  
непротизательности, 1284  
полезности, 783  
ранжирования вероятностей, 1112
- Принятие**  
решений в условиях неопределенности, 626  
решений людьми, 787  
решений последовательное, 799
- Приобретение**  
знаний, 64; 367  
надежное, 787
- Приращение**  
информации, 878; 882; 896
- Присваивание**, 210  
допустимое, 210  
полное, 210  
совместимое, 210
- Пробит-распределение**, 674; 708
- Проблема**  
выводимого окружения, 455  
индукции, 867  
исследования, 140  
непредвиденных ситуаций, 140; 143  
неразрешимая, 73

- окружения, 454; 497  
останова машин Тьюринга, 384  
отсутствия датчиков, 139  
поиска решения, 44  
полуразрешимости, 384  
понимания языка, 54  
представительного окружения, 455; 516  
разума и тела, 1258  
распространения последствий, 456  
совместимости, 139  
спецификации, 455; 518; 581; 622; 1255  
существования полной процедуры  
доказательства, 409  
трудноразрешимая, 73  
удовлетворения ограничений, 210
- Пробуксовка, 166
- Проведение  
метарассуждений, 269  
рассуждений, 58  
формальных юридических рассуждений, 74
- Проверка, 427  
вхождения, 387; 403  
значимости, 882  
перекрестная, 883; 990; 1008  
перекрестная с исключением одного  
примера, 883  
по моделям, 292  
показателя интеллекта, 59; 73  
предварительная, 219  
схемы, 373  
терминального состояния, 242  
цели, 114; 144; 211; 523; 1057
- Программа  
агента, 91
- Программа  
Absolver, 171  
Advice Taker, 58; 63  
Analogy, 59  
Aura, 428  
BKG, 275  
Bridge Baron, 270  
Chess 4.5, 146  
Chill, 1091  
Chinook, 266; 279  
DarkThought, 274  
Deep Blue, 69; 264; 265  
Dendral, 62  
EQP, 427  
Fritz, 265  
Geometry Theorem Prover, 57  
GIB, 267  
Go4++, 267  
Goemate, 267  
GPS, 57  
Kaissa, 273  
Logic Theorist, 56; 334  
Logistello, 266  
LT, 56
- MacHack 6, 273  
Mycin, 64  
Neurogammon, 1032  
Otter, 423  
Prolog, 401  
Proverb, 70  
PTTP, 425  
Remote Agent, 69; 105; 613  
Saint, 59  
Shrdlu, 60  
Student, 59  
TD-Gammon, 1040  
автоматического доказательства  
геометрических теорем, 57  
автоматического доказательства теорем, 434  
агента, 77; 91; 103  
логик-теоретик, 56  
общего решателя задач, 57  
планирования автономная бортовая, 69  
проверки моделей Spin, 428  
частичная, 1042  
шахматная ITEP, 272  
шахматная NSS, 272
- Программирование  
агентно-ориентированное, 105  
генетическое, 203  
динамическое, 200; 407; 730; 856; 1058  
динамическое адаптивное, 1015  
динамическое непоследовательное, 708  
индуктивное логическое, 917; 939; 942  
линейное, 189; 202; 213; 847  
логическое, 312; 375; 399; 401; 433  
логическое в ограничениях, 408; 433  
логическое табулированное, 407; 432; 433  
множества ответов, 488  
объектно-ориентированное, 52  
функциональное, 433
- Продление  
причинной связи, 602
- Продолжение, 405
- Продукция, 94
- Проект  
Deep Thought, 264  
Linguistic String Project, 1094  
Shakey, 58  
Soar, 68  
компьютера пятого поколения, 65  
эвристического программирования, 63
- Проектирование, 453  
агента, 839  
механизма, 840; 851  
молекулы белка, 122
- Проекция  
ортогональная масштабированная, 1145  
перспективная, 1144; 1156
- Прозрачность  
ссылочная, 468

- Проигрыш  
наихудший возможный, 789
- Произведение  
точечное, 679
- Проникновение, 199
- Пропозиционализация, 383
- Проставление  
обратных отметок, 235
- Простота  
конструирования, 331
- Пространство  
версий, 897; 908; 909  
гипотез, 868; 904  
доверительных состояний, 145  
занятое, 1210  
конфигураций, 1207; 1209  
рабочее, 1208  
свободное, 1210  
состояний, 114; 123; 144; 209  
состояний метауровневое, 167  
состояний объективно-уровневое, 167
- Протагор, 1093
- Протез  
мозга, 1261; 1275
- Противодействие  
языка синонимии, 1116
- Противоречие, 424
- Профиль  
стратегии, 841
- Процедура  
доказательства, 44  
логического вывода, 368  
логического вывода полная, 413
- Процесс, 462  
инженерии знаний, 367  
марковский, 721  
обнаружения края Кэнни, 1153  
принятия решений комплексный, 200  
принятия решений марковский, 46; 817; 855; 1011  
принятия решений марковский в частично наблюдаемой среде, 832; 857
- Процессор  
текстовый Microsoft Word, 1121
- Псевдокод, 1298
- Псевдоэксперимент, 1019
- Психология  
когнитивная, 50  
народная, 502; 1274  
обывательская, 50  
экспериментальная, 37; 49
- Психофизика, 1184
- Путь, 114; 144; 566  
кратчайший, 150  
критический, 566  
опорный, 1221
- Пятно  
светочувствительное, 1178
- Пьеса  
R.U.R., 1237
- P**
- Работы  
разведочные, 799
- Равновесие, 1017  
Байеса–Нэша, 851  
детализированное, 692  
доминантных стратегий, 842  
максиминное, 847  
Нэша, 843; 847
- Радар, 46
- Разбиение  
на лексемы, 1123
- Разбор  
синтаксический, 1056
- Развертывание  
литеративное, 201  
состояния, 122
- Раздача  
карт в покере, 656
- Разделение  
переменных, 648
- Разделитель  
линейный, 981; 991  
линейный оптимальный, 993
- Разложение  
в ряд Тейлора, 1202
- Разметка  
линий, 1164
- Разновидность  
естественная, 447
- Разработка  
генетических алгоритмов, 61
- Разрешение  
ссылок, 1085
- Разум, 35; 1272  
как физическая система, 41
- Ракурс, 1162
- Рамsey Ф., 45
- Ранг  
обратный, 1115
- Рандомизация, 76; 95
- Раскрашивание  
графа, 234  
карты, 234
- Расписание, 567
- Распознавание  
введенных вручную цифр, 1168  
лица, 1172  
объекта, 1154  
объектов, 1168  
плана, 610

- речи, 67; 734; 758; 773; 774; 1051  
 рукописного текста, 1168  
 рукописных цифр, 1170  
 с учетом характеристик, 1171  
 с учетом яркости, 1171  
 символов оптическое, 1051
- Распределение**  
 априорных вероятностей, 631  
 априорных вероятностей  
     оптимистическое, 1024  
 априорных вероятностей равномерное, 950  
 вероятностей, 268; 675  
 вероятностей полное совместное, 631;  
 653; 660  
 вероятностей совместное, 631  
 гауссово линейное, 672; 708  
 гауссово смешанное, 963; 966; 967  
 гауссово условное, 674  
 Дирихле скрытое, 1135  
 каноническое, 669  
 линейное гауссово, 738  
 с широким хвостом, 202  
 связи  $\vee$  по  $\wedge$ , 412  
 смешанное, 963  
 сопряженное априорных вероятностей, 958  
 стационарное, 692
- Распространение**  
 единичных выражений, 317  
 обратное, 986  
 ограничений, 59  
 ограничения, 220  
 оценок степени уверенности, 710  
 пределов, 224
- Рассел Б., 42; 53; 54; 56  
 Рассел С.Дж., 449  
 Рассогласование, 1159; 1160
- Расстояние**  
 измеряемое в городских кварталах, 168  
 манхэттенское, 168  
 по прямой, 155
- Рассуждение**  
 управляемое данными, 315  
 управляемое целями, 315
- Рассуждения**  
 межпричинные, 702  
 пространственные, 502  
 психологические, 502  
 управляемые целью, 270
- Расширение**  
 одинарное, 256  
 предиката, 904  
 существующих правил грамматики, 1066  
 теории умолчаний, 490
- Расщепление**  
 символа, 550
- Рациональность**, 35; 80; 105  
 вычислительная, 56; 1283
- идеальная, 40; 1283  
 ограниченная, 40; 1283
- Ребро**, 1059  
 полное, 1059
- Ребус**  
 числовой, 213
- Регистрация**  
 ограничения, 235
- Регрессия**, 896  
 линейная, 956
- Регулятор**  
 паровой машины, 52
- Режим**  
 разделения времени, 58  
 релейного управления, 1032
- Резерв**  
 времени, 567  
 минимальный, 569
- Революция**, 409; 429; 430; 944  
 бинарная, 412  
 единичная, 422  
 линейная, 423; 936  
 обратная, 934; 942  
 пропозициональная, 409  
 с входными высказываниями, 422  
 с линейным выходным выражением, 426  
 теории, 434
- Резольвента, 422; 934
- Резонанс**  
 магнитный функциональный, 47
- Результат**, 515; 516  
 внешний, 573  
 внутренний, 573  
 вторичный, 573  
 игры, 841  
 неявный, 456  
 оптимальный согласно принципу Парето, 842  
 отрицательный, 559  
 первичный, 573  
 положительный, 525
- Результаты**  
 дизъюнктивные, 584  
 знаний, 470  
 условные, 584
- Рейс М., 267
- Релевантность**, 298; 916; 941
- Репозитарий**  
 кода, 90
- Ресурс**  
 повторно применяемый, 567  
 потребляемый, 568
- Ресурсы**  
 в планировании, 611
- Речь**, 1085
- Решатель**  
 задач универсальный, 37
- Решение**, 112; 115; 144; 210; 530

- в планировании, 516  
 единоразовое, 780  
 задач, 62  
 задачи MDP, 1216  
 игры, 841  
 минимаксное, 244  
 надежное, 1217  
 оптимальное, 115  
 последовательное, 780  
 рациональное, 623; 778; 803
- Решетка**  
 обобщения, 389; 436  
 прямоугольная, 137
- Рисунок**  
 контурный, 1164
- Роббинс Г.**, 427
- Робинсон А.**, 431
- Робинсон Дж.А.**, 58
- Робинсон Дж.Э.**, 409; 416
- Робот**, 1188; 1237  
 Carmel, 1238  
 Helpmate, 1232  
 PUMA, 1237  
 Shakey, 105; 555; 561; 615; 1238  
 Sojourner, 1189  
 Unimate, 1237  
 голономный, 1194  
 мобильный, 1189  
 неголономный, 1194  
 программный, 85  
 шестиногий, 1225
- Робот-гуманоид**, 1189
- Робот-манипулятор**, 1188
- Робототехника**, 37; 749; 1189; 1237  
 когнитивная, 498
- Робот-футболист**, 107; 241
- Рождество**, 1256
- Розенблют А.**, 53
- Россия**, 771
- Рост**  
 сложности экспоненциальный, 44
- Рочестер Н.**, 55; 57
- Румельхарт Д.Э.**, 66
- Румыния**, 111; 210
- C**
- Саймон Г.Э.**, 37; 46; 51; 55; 264
- Самоанализ**, 73
- Самодвижение**, 1157
- Самолет**  
 беспилотный, 1234
- Самюэл А.Л.**, 55; 57; 265; 1040
- Санскрит**, 1093
- СБИС**  
 сверхбольшая степень интеграции, 146
- Сбор**
- информации, 81  
 информации с помощью датчиков, 614  
 информации с помощью датчиков автоматический, 592  
 информации с помощью датчиков активный, 592
- Сборка**  
 космических аппаратов, 613
- Свертка**, 1186
- Свертывание**  
 пространства версий, 912
- Свидетельство**, 625
- Свойства**  
 отражательные, 1146; 1162  
 проблемной среды, 86
- Свойство**  
 внешнее, 450  
 внутреннее, 450  
 как унарное отношение, 345  
 локальности, 329  
 марковости, 732; 770  
 марковское, 817
- Связка**  
 “исключительное ИЛИ”, 298  
 логическая, 55; 295; 352
- Связь**  
 IS-A, 500  
 взаимно исключающая, 538  
 инверсная, 479  
 обратная тактильная, 1240  
 овеществленная, 479  
 причинная, 530
- Сглаживание**, 1149; 1151  
 Гуда-Тьюринга, 1135  
 нулевых результатов, 1104  
 оперативное, 736  
 с добавлением единицы, 1104  
 с линейной интерполяцией, 1104  
 с удалением путем интерполяции, 1135
- Сегментация**, 445; 1105; 1153  
 изображения, 1169  
 речи, 765
- Селфридж О.Г.**, 55
- Семантика**, 72; 1048  
 композиционная, 1072  
 логики первого порядка, 347  
 логическая, 332  
 предпочтений, 1095  
 пропозициональной логики, 297  
 языка, 290
- Семейство**  
 языков Planner, 64
- Семинар**  
 Дартмутский, 55; 56
- Семиотика**, 1093
- Сетчатка**, 1141
- Сеть**

- байесовская, 68; 660  
байесовская гибридная, 672  
байесовская динамическая, 718; 746; 771; 772; 817; 836  
беспроводная, 1195  
задач, 556  
задач иерархическая, 518; 570  
качественная вероятностная, 711; 793  
многосвязная, 682  
нейронная, 55; 59; 65; 267; 976  
нейронная многослойная, 62; 990  
нейронная многослойная с прямым распространением, 999  
нейронная однослойная, 980  
нейронная с прямым распространением, 979  
нейронная с радиальной базисной функцией, 1003  
односвязная, 681  
принятия решений, 661; 778; 795; 798; 808; 809; 836  
принятия решений динамическая, 836; 856  
различительная, 896  
рекуррентная, 979; 1004  
с прямым распространением, 979  
семантическая, 477; 499  
Хопфилда, 1004
- Сжатие, 826; 1028  
Силлогизм, 38; 333; 429  
Символ  
    константный, 349; 351; 374  
    начальный, 1297  
    нетерминальный, 1049; 1050; 1297  
    предикатный, 349; 374  
    пропозициональный, 295  
    равенства, 357  
    терминальный, 1048; 1297  
    функциональный, 349; 351; 374  
    чистый, 317
- Синапс, 48  
Синоним, 1116  
Синтаксис, 64; 290  
    логики первого порядка, 347  
    логический, 332
- Синтез, 427; 1051  
    алгоритмов, 428  
    дедуктивный, 428  
    программ автоматизированный, 614
- Сирл Дж.Р., 1259; 1261; 1273
- Система  
    сCHUGIN, 708  
    Core Language Engine, 1094  
    Deep Blue, 273  
    Deep Thought, 273  
    Fastus, 1123  
    GPS дифференциальная, 1191  
    HipNav, 70  
    Pengi, 616
- PTTP, 434  
R1, 65  
Shrdlu, 64  
TD-Gammon, 1032  
Wordnet, 1138  
World Wide Web, 471  
Xeon, 398  
десятичная Дьюи, 444  
динамическая, 771  
жизнеобеспечения интеллектуальная, 1190  
истинностно-функциональная, 707  
компьютерного зрения Alvinn, 69  
координат, 1155  
логическая на основе правил, 700  
логического программирования, 429  
локально структурированная, 666  
многотельная, 1190  
мультиагентная, 840  
на основе правил, 1254  
обеспечения истинности, 235  
обозначений арифметических, 38  
обозначений инфиксная, 361  
обозначений логических, 38  
обозначений префиксная, 361  
операционная Microsoft Windows, 707  
основанная на знаниях, 1027  
перевода Taut-Meteo, 1138  
планирования путешествий, 120  
планирования регрессивная, 43  
поддержки истинности, 492; 501; 1274  
поддержки истинности на основе обоснований, 492  
поддержки истинности на основе предположения, 493  
позиционирования глобальная, 1191  
продукционная, 380; 398; 429; 431  
разреженная, 666  
с разомкнутой обратной связью, 113  
символическая физическая, 57  
телеежка-шест, 1032  
технического зрения, 37; 39; 49; 59; 234  
технического зрения активная, 1255  
формальная грамматическая, 1049  
экспертная, 63; 398; 495; 699; 804; 808; 943  
экспертная Meta-Dendral, 897  
экспертная в условиях неопределенности, 68  
экспертная впервые созданная, 63  
экспертная медицинская, 70; 710; 1268  
экспертная на основе Prolog, 401  
экспертная нормативная, 68  
экспертная первая коммерческая, 65
- Ситуационность, 68  
Ситуация, 451; 779  
Скелет  
    пространства конфигураций, 1214
- Скелетирование, 1214  
Скиннер Б.Ф., 53  
Сколемизация, 382; 411

- Скорость  
обучения, 1017
- Скрещивание, 183
- Следствие, 291  
логическое, 291  
логическое обратное, 936
- Слияние  
байесовских моделей, 1110  
структур, 1124
- Словарь, 1054  
электронный, 61
- Слово, 1047; 1048  
“tomato”, 763  
запретное, 1119  
сложное, 1123
- Словосочетание, 1049  
глагольное, 1049; 1055  
именное, 1049; 1055  
предложное, 1055  
указательное, 1078  
флюентное, 1079
- Сложность  
алгоритмическая, 898  
временная, 126; 144  
выборочная, 891  
данных, 396; 432  
колмогоровская, 898  
оценки ожидаемых минимаксных  
значений, 261  
пространственная, 126; 144
- Служба  
перевода Systran, 1125
- Случай  
обусловливающий, 663
- Смещение  
декларативное, 927
- Смит А., 45
- Событие, 440; 675  
атомарное, 629  
вневременное, 462  
дискретное, 462  
обобщенное, 460
- Совершенство, 81
- Совместимость, 904  
дуги, 220; 222  
пути, 222; 234  
с гипотезой, 868  
узла, 222
- Совокупность, 446  
текстов, 1102
- Согласование  
с учетом деформации, 1173  
с шаблоном, 394
- Соединенные Штаты Америки, 69; 217; 798
- Сожаление, 788
- Создание  
мультимножеств, 897
- новых предикатов, 937
- Сознание, 47; 1261; 1262
- Сократ, 38
- Сокращение, 487
- Соломонов Р.Дж., 55
- Сома, 48
- Сонет, 1256
- Сообщение  
о погоде, 1125
- Соревнования  
Robocup, 1238
- Составление  
карты, 1203  
кроссворда, 237
- Состояние, 124; 463  
внутреннее, 96  
динамическое, 1193; 1220  
доверительное, 141; 589; 627; 724; 833; 1196  
кинематическое, 1193  
мира, 111  
мозга, 1259  
наиболее вероятное, 1216  
начальное, 113; 144; 211; 242; 1056  
повторяющееся, 137  
психическое, 1258  
текущее в локальном поиске, 174  
терминальное, 242
- Софтбот, 85
- Сохранение  
истинности в логическом выводе, 293
- Сочетание  
гауссовых распределений, 761  
двухсловное, 1116
- Сочленение  
призматическое, 1193
- Спам, 1139
- Список, 363  
добавления, 515  
закрытый, 138  
одновременных действий, 607  
открытый, 138  
позиций, 1119  
решений, 892  
связывания, 358  
субкатегоризации, 1069  
удаления, 515
- Способ  
представления, 1111  
сведения, 1295
- Способности  
сенсорные, 74  
человека, 35
- Способность  
мутагенная, 938  
отражательная, 1147  
порождающая, 1049; 1071

- Спуск  
градиентный, 180
- Спутник  
Земли искусственный, 61
- Среда, 75  
ведения игры, 278  
вождения, 83  
детерминированная, 86; 113  
динамическая, 87  
дискретная, 87; 113  
доступная, 86  
искусственная, 84  
конкурентная, 88  
кооперативная, 88  
мультиагентная, 88; 466; 605  
наблюдаемая, 113  
недетерминированная. См. среда  
стохастическая  
недоступная, 86  
непрерывная, 87  
неэпизодическая. См. Среда  
последовательная  
одноагентная, 88  
однократная, 87  
полностью наблюдаемая, 86; 816; 831  
полудинамическая, 87  
последовательная, 86; 87  
проблемная, 82; 83  
статическая, 87  
стохастическая, 86  
стратегическая, 87  
функционирования такси, 83  
частично наблюдаемая, 86; 831  
эпизодическая, 87
- Средства  
цифровой связи, 1195
- Средство  
автоматизированного формирования  
рассуждений, 423  
автоматического доказательства теорем, 423  
общения, 343  
представления, 343  
 проверки доказательства, 427  
транспортное автономное, 69  
транспортное воздушное  
автоматическое, 1189  
транспортное наземное автоматическое, 1189  
транспортное подводное автономное, 1189  
формирования рассуждений  
сократовское, 427
- Срез  
временной, 719
- Срок  
существования агента, 92
- СССР, 272
- Стабильность  
контроллера, 1222
- Стандарт  
Semantic Web, 496  
XML, 496
- Стандартизация  
отличий, 437  
отличия, 386  
переменных, 411
- Станок  
жаккардов, 52  
ткацкий, 52  
ткацкий Жаккарда, 1237
- Статья  
научная советская, 61
- Стационарность, 890  
предпочтений, 820
- Стек, 130  
контрольный, 404  
пространств фокусов, 1089
- Стекло  
спиновое, 1002
- Степень  
подтверждения, 654  
свободы, 1192  
свободы управляемая, 1194  
свободы эффективная, 1194  
уверенности, 624; 637
- Стереограмма  
со случайными точками, 1182
- Стереоданные  
бинокулярные, 1156; 1158; 1179; 1182
- Стимул, 50
- Стирка  
белья, 1101
- Стоимость  
информации, 808; 810; 814; 834  
материала, 253  
оптимального решения подзадачи, 172  
поиска, 126  
поиска суммарная, 126  
полней информации, 800  
пути, 114; 144; 211  
решения, 126  
уровневая, 540  
этапа, 115
- Стратегия, 243; 818; 855  
“взаимно гарантированного  
уничтожения”, 849  
доминантная, 842  
игры, 840  
нестационарная, 820  
оптимальная, 818  
поиска, 122  
правильная, 821; 1042  
с наименьшим вкладом, 527  
с преимущественным использованием  
единичных выражений, 422  
смешанная, 841

- стационарная, 820  
чистая, 840
- Страхование**  
жизни, 788  
от автомобильной аварии, 788
- Стремление**  
к риску, 787
- Строка**  
в логике, 468
- Структура**  
агента, 91  
данных узла, 124  
словосочетания, 1049
- Субкатегоризация**, 1069  
глагола, 1068
- Субъективизм**, 634
- Суждение**  
субъективное, 710
- Суждения**  
о вероятностях, 668
- Сулавеси**, 229
- Сумма**  
бесконечных геометрических рядов, 821  
квадратичных ошибок, 956  
квадратов разностей, 1157
- Супермен**, 343; 467
- Супермозг**, 45
- Существительное**  
исчисляемое, 450  
неисчисляемое, 450
- Схема**  
GLIE, 1023  
в генетическом алгоритме, 186  
действия, 515  
жадная в пределе бесконечного исследования, 1023  
логическая ациклическая, 329  
логическая циклическая, 329  
математической индукции, 417  
последовательная логическая, 325  
правила, 1071
- Сходимость**  
итерации по значениям, 826
- США**, 50; 773; 811; 995; 998; 1267
- T**
- Таблица**  
“действие–полезность”, 797  
истинностная, 297; 335  
поисковая, 990  
транспозиций, 251  
треугольная, 615  
условных вероятностей, 663
- Табуляция**  
функции агента, 76
- Тавтология**, 302
- Такси**, 83
- автоматизированное, 83; 102  
в Афинах, 658
- Таксономия**, 444; 474; 496
- Талос**, 1237
- Тарский А.**, 43; 427
- Тасмания**, 228
- Тезис**  
Чёрча–Тьюринга, 44
- Тексел**, 1161
- Текст**  
анкера, 472
- Текстура**, 1156; 1161
- Телевидение**, 1046
- Телескоп**, 713  
космический Хаббл, 212; 228; 564; 613
- Тело**  
выражения, 312; 400  
клетки, 48  
твердое трехгранное, 1166
- Теннис**  
настольный, 74  
парный, 605; 609
- Теорема**, 360  
дедукции, 302  
Нэша, 843  
о неполноте, 44; 409; 417; 1251  
о полноте, 409  
о раскраске карты четырьмя цветами, 1253  
представления, 793  
резолюции базовая, 417  
резолюции основная, 311  
сходимости перцентрона, 59  
Эрбрана, 394; 419; 430
- Теоретик**, 66
- Теория**  
Болдуина, 186  
вероятностей, 45; 67; 346; 624; 807  
возможностей, 711  
вычислительного обучения, 889; 894; 896; 898; 1003  
Демпстера–Шефера, 700; 711  
жидкостей, 502  
игр, 46; 815; 839  
игр обратная, 851  
информации, 877; 897  
Ламарка, 186  
многоатрибутной полезности, 790; 808; 809  
моделей, 375  
мышления, 37  
нормативная, 787  
описательная, 787  
оптимального управления, 202  
оптимальности, 1095  
оптимизации, 202  
подтверждения, 42  
полезности, 626; 807  
принятия решений, 807

- равномерной сходимости, 898  
 решений, 46; 67; 627  
 решеток, 427  
 риторической структуры, 1096  
 синтаксическая, 468  
 ссылок, 43  
 стоимости информации, 798; 1255  
 суперструн, 460  
 управления, 52; 53; 104; 554; 1002;  
 1032; 1222  
 управления адаптивная, 1014  
 функциональной спецификации, 1273  
 ценности информации, 1282  
 чисел, 943
- Терм**  
 базовый, 353; 381  
 в логике, 351  
 квантифицированный, 1076  
 с кванторами, 1078  
 сложный, 374
- Термостат, 52
- Тесауро Г., 267; 1032
- Тест**  
 Тьюринга, 36; 39; 72; 73; 1046; 1250  
 Тьюринга полный, 36
- Техника**  
 вычислительная, 51
- Тинсли М., 266
- Тип**  
 соединения, 1167
- Точка**  
 ближайшая соседняя, 1173; 1175  
 выбора, 403  
 критическая, 321  
 разбиения, 884  
 схода, 1145  
 фиксированная, 314; 393; 826
- Точность, 1114  
 функции оценки, 252
- Тракт**  
 голосовой, 762
- Трактат**  
 Органон, 333; 496
- Транзитивность**  
 предпочтений, 781; 782
- Транспорт**  
 грузовой воздушный, 518
- Трассировка**  
 луча, 1147
- Требования**  
 к памяти, 133
- Тренажер**, 85
- Трещина**, 1166
- Троянский П.**, 1137
- Тупик**, 192
- Турок**, 271
- Тьюринг А., 36; 44; 54; 55; 57; 241; 1127;  
 1250; 1253; 1256; 1263; 1275; 1287
- Y**
- Уайтхед А.Н., 54  
 Уатт Дж., 52  
 Убеждения, 466  
 Убыточность  
 стратегии, 828
- Углубление  
 итеративное, 134
- Угол**  
 наклона, 1155  
 поворота, 1155
- Удаление**  
 асбестовых изделий, 789  
 кванторов всеобщности, 411  
 потенциальных гипотез, 908  
 условий, 907
- Удовлетворение, 46  
 ограничений, 59
- Узел**, 124  
 дерева поиска, 123  
 жеребьевки, 258; 796  
 значения, 796  
 листовой, 125  
 поисковый, 122  
 полезности, 796  
 принятия решений, 796  
 родительский, 124
- Уилkins Д.Э., 270
- Уильямс Р., 811
- Университет**  
 СМУ, 69; 273; 773  
 Йельский, 64  
 Карнеги-Меллона, 55; 56  
 Кембриджский, 50  
 Лейпцигский, 49  
 Пенсильянский, 51  
 Принстонский, 55  
 Станфордский, 56; 58; 62; 63; 375
- Универсум**  
 Эрбрана, 418; 430; 431
- Унификатор**, 386  
 наиболее общий, 387; 389; 419; 436
- Унификация**, 380; 386; 428  
 с учетом равенства, 421
- Уолтер Г., 1237
- Уоррен**  
 Д.Г.Д., 404
- Уотсон Дж., 49
- Упорядочение**  
 конъюнктов, 395  
 переменных, 217
- Упорядочиваемость**, 782
- Управление**

- движением в поперечном направлении, 1179  
 движением в продольном направлении, 1179  
 нечеткое, 705  
 оптимальное стохастическое, 53  
 реактивное, 1225
- Упрощение**  
 синтаксическое, 361; 402
- Уравнение**  
 Беллмана, 824  
 Витерби, 1105  
 временной разности, 1017  
 диофантово, 233  
 дифференциальное, 1220  
 Ковальского, 401  
 фильтрации рекурсивное, 833
- Уровень**  
 алгоритмический, 1229  
 в графе планирования, 536  
 знаний, 285; 333  
 исполнительный, 1229  
 реактивный, 1229  
 реализации, 285
- Усвоение**, 1053
- Усиление**, 885; 896
- Условие**  
 завершения, 1217  
 монотонности, 199  
 преемственности, 199
- Усовершенствование**  
 стратегии, 829
- Усреднение**  
 по прогнозам, 263
- Устойчивость**  
 динамическая, 1194  
 статическая, 1194
- Устранение**  
 импликации, 410  
 неоднозначности, 1052  
 переменной, 678; 681; 706; 708; 753
- Устройство**  
 Heath Robinson, 51  
 гомеостатическое, 1002  
 протезное, 1189
- Уступ**, 177
- Утверждение**  
 логическое, 358
- Утилитаризм**, 43
- Утка**  
 механическая, 1237
- Уточнение**, 905; 906
- Учет**  
 неопределенности, 624
- Учитель**, 866
- Ф**
- Факт**, 312
- Фактор**
- определенности, 702; 710  
 устранения переменной, 679
- Факторизация**, 308
- Фальстарт**, 156
- Фейгенбаум Э.А.**, 62; 72
- Фельдман Дж.**, 72
- Ферма П.**, 45; 654
- Фертильность**, 1131
- Фигура**  
 речи, 1083
- Физика**  
 качественная, 449; 501
- Фиксация**, 1160
- Филон из Мегары**, 334
- Философия**, 40; 104  
 разума, 1272; 1275
- Фильм**  
 А.И., 1271  
*The Matrix*, 1270  
*The Terminator*, 1270
- Фильтр**  
 Калмана, 718; 742; 747; 771; 1200  
 Калмана переключательный, 745  
 Калмана расширенный, 1202  
 с гауссовой характеристикой, 1149  
 частиц, 1238
- Фильтрация**, 724; 727  
 калмановская, 738; 771  
 частиц, 756; 772
- Флюентный**, 452
- Фокус**, 1146  
 внимания, 1089  
 расширения, 1158
- фон Гельмгольц Г.**, 49
- фон Кемпелен В.**, 271
- фон Нейман Дж.**, 45; 53
- Фонд**  
 Рокфеллера, 1137
- Фонема**, 759
- Форма**, 1155  
 3-CNF, 308  
 k-CNF, 308
- Бэкуса-Наура**, 1049; 1297
- квазилогическая**, 1076; 1095
- Ковальского**, 340  
 нормальная импликативная, 340  
 нормальная конъюнктивная, 308; 410  
 нормальная Хомского, 1110  
 представления синтаксиса, 290  
 промежуточная, 1075  
 хорновская, 312; 334
- Формирование**  
 вознаграждения, 1041  
 выборок логическое, 709  
 выборок последовательное повторное, 772  
 выборок с учетом важности, 709  
 изображения, 1180

- логических выводов автоматическое, 36  
ослабленной задачи автоматическое, 170  
рассуждений в условиях неопределенности, 68  
рассуждений на основе модели, 366  
рассуждений по умолчанию, 699  
состояния, 122
- Формирователь**  
выборок Гиббса, 693; 709
- Формула**  
VPI, 801
- Формулировка**  
задачи, 111  
инкрементная, 118; 211  
полного состояния, 119; 176  
с полным состоянием, 211  
цели, 111; 582
- Фортепьяно**  
логическое, 334
- Фотограмметрия**, 1182
- Фотометрия**, 1146
- Фрагмент**  
речи, 1047  
речи неоднозначный, 1052
- Фраза**  
сложная, 1124
- Франкенштейн**, 1270
- Фреге Г.**, 43; 374; 430
- Фрейм**, 64; 500; 760
- Фrekвентизм**, 633
- Функционализм**, 105; 1263; 1273  
Аристотеля, 1272
- Функциональность**  
истинностная, 701
- Функция**, 345  
“действие–значение”, 1011  
“деления на два”, 826  
Den, 468  
softmax, 1034  
агента, 76; 91; 818  
активации, 977  
активации пороговая, 980  
базисная, 1028  
булева, 979  
вознаграждения, 242; 626; 835; 855; 1013  
вычислимая, 44  
Гашнига, 206  
доверительная, 703  
единиц измерения, 448  
значения, 789  
значения аддитивная, 794  
исключительного ИЛИ, 1007  
исследования, 1024; 1026  
линейная взвешенная, 253  
линейно разделимая, 981; 999  
мажоритарная, 872; 980  
навигации, 1216
- определения преемника, 113; 198; 211; 242; 1056  
определения четности, 872  
оценки, 154; 241; 252; 1028  
оценки линейная, 174  
плотности вероятностей, 631; 1294  
полезности, 99; 242; 779; 783; 817; 823; 1011; 1029  
полезности мультиплективная, 795  
полезности порядковая, 789  
полностью определенная, 349  
пригодности, 182  
сжатия, 826  
сколемовская, 411; 430  
целевая, 53; 175; 210  
эвристическая, 154; 513  
эвристическая допустимая, 157  
эвристическая на основе манхэттенского  
расстояния, 168  
эвристическая определения расстояния по  
прямой, 155  
ядерная полиномиальная, 995
- Футбол**  
робототехнический, 616; 1235
- X**
- Характеристика, 1028  
акустическая, 760  
состояния, 173
- Хебб Д.О., 55; 59
- Хекерман Д., 810
- Хемпель К., 42
- Хивуд П., 1253
- Химик-аналитик, 63
- Химия, 63
- Хинтон Дж.Э., 66
- Холостяк, 445
- Хомский Н., 51; 54
- Хребет, 177
- Хронология, 498
- Хэш-таблица, 388
- Ц**
- Целенаправленность, 1273; 1274
- Цель**, 97; 111; 144; 515  
логического вывода, 358  
обучения, 167  
отрицаемая, 414
- Ценность**  
вычислений, 1282  
стратегии, 1035
- Цепь**  
Маркова, 692  
марковская, 721
- Цузе К., 51

**Ч**

- Чапек К., 1237; 1270  
 Чарняк Ю., 64  
 Частота  
     дискретизации, 760  
 Часть, 445  
 Чатбот, 1250  
 Человек  
     пещерный, 914  
 Чередование  
     поиска и действия, 143  
 Чёрч Э., 44  
 Число  
     конспиративное, 274  
     натуральное, 361

**III**

- Шаг  
     итерации, 825  
 Шанкар Н., 427  
 Шанс  
     на победу, 252  
 Шарнир  
     поворотный, 1193  
 Шахматы, 52; 60; 136; 146; 241; 271  
     с контролем времени, 87  
 Шахтер Р.Д., 789  
 Шашки, 57; 106; 275  
 Шелински Р., 1156  
 Шенк Р.К., 64  
 Шеннон К.Э., 55; 241  
 Шеффер Дж., 266  
 Шиккард В., 41  
 Шимпанзе, 1046  
 Ширина  
     гипердерева, 236  
     дерева, 236  
     индуцированная, 236  
 Школа  
     бизнеса гарвардская, 788  
     мегарская, 334; 496  
     стоическая, 334; 496  
 Шортлифф Э.Г., 63  
 Шоу  
     игровое, 785  
 Штраф, 103  
 Шум, 875; 912; 927; 945
- Э**
- Эванс Т.Г., 73  
 Эволюция, 74; 185; 343  
     машин, 61  
 Эвристика, 154; 198  
     допустимая, 525  
     единичного выражения, 317
- максимального уровня, 540  
 множественного уровня, 540  
 независимая от проблемной области, 513  
 плавания вдоль берегов, 1217  
 с минимальным количеством оставшихся значений, 217  
 с минимальными конфликтами, 227; 235  
 с наиболее ограниченной переменной, 234; 395; 536  
 с наименее ограничительным значением, 218  
 с пустым списком удаления, 527  
 составная, 171  
 степенная, 218  
 чистого символа, 317  
 Эдинбург, 942; 1239  
 Эдмондс Д., 55  
 Эйнштейн А., 34  
 Эквивалент  
     определенности, 787  
 Эквивалентность  
     логическая, 302  
     с точки зрения логического вывода, 382  
 Экерт Дж., 51  
 Экземпляр  
     схемы, 186  
     типичный, 447  
 Экономика, 45; 104; 784  
 Экономия  
     усилий, 624  
 Эксперимент  
     мозг в колбе, 1260  
 Экспериментатор, 66  
 Электродвигатель, 1195  
 Электростанция  
     атомная, 713  
 Электроэнцефалограф, 47  
 Элемент  
     логический в логике, 370  
     обучающий, 864  
     проблемной области, 347  
     производственный, 864  
     с зарядовой связью, 1143  
     скрытый, 979  
 Эмпиризм, 41  
 Эмуляция  
     отжига, 180; 202  
 Эпифеноменальный, 1262  
 Эпсилон-шар, 890  
 Эрбран Ж., 383; 430  
 Эскимос, 344  
 Этап  
     избыточный, 602  
     условный, 585  
 Эффект  
     Болдуина, 186

горизонта, 255  
Эффективность  
вычислительная, 330  
ЭЭГ  
электроэнцефалограф, 47

## Ю

Юм Д., 41  
Юнг Т., 1148

## Я

Язык, 1046  
ADL, 517; 555  
ALisp, 1231  
Buffalo, 1100  
CES, 1231  
CLIPS, 432  
Datalog, 392; 406; 432  
Fortran, 58  
Golog, 1231  
GRL, 1230  
HTML, 1123  
k-DL, 892  
k-DT, 892  
Lisp, 351; 469; 892; 1230  
Micro-Planner, 431  
MRS, 409  
Perl, 469  
Planner, 431  
Prolog, 64; 401; 555; 934  
Qlisp, 434  
SGML, 1123  
SQL, 484  
WAM, 404

XML, 1123  
английский, 61; 74; 1125  
английский компаний Caterpillar, 1125  
быстрой разработки прототипов, 401  
высокого уровня Lisp, 57  
естественный, 39; 343; 1048  
запросов, 1110  
итальянский, 1125  
композициональный, 342  
мышления, 343  
ограничений, 212  
поведения, 1230  
португальский, 915  
представления знаний, 284; 332; 341  
программирования, 342  
программирования Ada, 52  
программирования Plankalkül, 51  
программирования Prolog, 1093  
программирования для искусственного  
интеллекта, 57  
программирования робототехнический, 1230  
продукционных систем Ops-5, 431  
промежуточный, 1126  
псевдокода, 91  
робототехнический универсальный, 1230  
русский, 61  
санскрит, 495  
формальный, 1048  
шведский, 74

Яма  
бездонная, 286  
Япония, 65  
Яркость, 1146

*Научно-популярное издание*

**Стюарт Рассел, Питер Норвиг**

**Искусственный интеллект: современный подход  
2-е издание**

Литературный редактор *И.А. Попова*

Верстка *О.В. Линник*

Художественный редактор *С.А. Чернокозинский*

Корректоры *З.В. Александрова, Л.А. Гордиенко,*

*О.В. Мишутина, Т.А. Корзун*

Издательский дом “Вильямс”  
127055, г. Москва, ул. Лесная, д. 43, стр. 1

Подписано в печать 12.04.2007. Формат 70x100/16.

Гарнитура Times. Печать офсетная.

Усл. печ. л. 114,8. Уч.-изд. л. 106,1.

Доп. тираж 2000 экз. Заказ № 0000.

Отпечатано по технологии CtP  
в ОАО “Печатный двор” им. А.М. Горького  
197110, Санкт-Петербург, Чкаловский пр., 15.